이학박사 학위논문

A Higher-order Finite-Difference Scheme for Equity Swaps and Heterogeneous Computing with OpenCL

Equity Swaps에 대한 고차수렴 유한차분법과 OpenCL을 이용한 Heterogeneous 컴퓨팅

2013년 8월

서울대학교 대학원
협동과정 계산과학전공
추형석

A Higher–order Finite–Difference Scheme for Equity Swaps and Heterogeneous Computing with OpenCL

Equity Swap에 대한 고차 유한차분법과 OpenCL을 이용한 Heterogeneous 컴퓨팅

지도교수 신동우

이 논문을 이학박사 학위논문으로 제출함
2013년 6월

서울대학교 대학원
협동과정 계산과학전공
추형석

추형석의 이학박사 학위논문을 인준함
2013년 6월

위 원 장: 이재진 (인)
부 위 원 장: 신동우 (인)
위 원: 기호삼 (인)
위 원: 김임범 (인)
위 원: 전영목 (인)

Abstract

A Higher–order Finite–Difference Scheme for Equity Swaps and Heterogeneous Computing with OpenCL

Hyoungseok Chu
Interdisciplinary Program in Computational Science and Technology
The Graduate School
Seoul National University

A nine-point compact finite difference scheme with fourth-order convergence is proposed for an equity swap model. In order to derive a compact scheme for the equity swap model, a special treatment is necessary to remove the mixed derivative term so that the resulting scheme is of fourth-order convergence as well as compactness. A suitable coordinate transformation is proposed to eliminate the mixed derivative term successfully. The resulting algorithm is shown to be a fourth order convergent scheme. Various examples confirm the validity of the proposed scheme.

Since most of linear solvers consist of basic linear algebra subroutines (BLAS), we optimize computational performance by distributing a subroutine into CPU and GPU with some splitting ratio. We present this splitting ratio by means of a min–max problem concerning with computational times. Computational times for both CPU and GPU are estimated as polynomial functions based on their capabilities. BLAS saxpy, sgemv, and sgemm are implemented in OpenCL and we verified our min–max model with actual heterogeneous computing results.

Keywords : Higher-Order Finite-Difference, Equity Swaps, Heterogeneous Computing, OpenCL, BLAS, Min–Max Model
Student Number : 2009-20453

Contents

List of Figures

List of Tables

Part I

A Higher–Order Finite–Difference Scheme for Equity Swaps

Chapter 1

Introduction

Recent global financial crises have been stimulating bank managers to understand interest rates thoroughly and to manage cautiously. For this, researchers need to develop more precise interest rate models in one direction, while they should calculate them more accurately in another direction. In this thesis, we are particularly interested in developing a highly efficient numerical scheme for a PDE equity swap model.

Several interest rate models have been developed actively and a growing attention has been paid year by year. For instance, see [9, 10, 13, 15, 18, 19, 24, 27–30, 33], and related references therein. From the PDE point of view, an equity swap follows the characteristic of Black-Scholes equation so that one can evaluate a pricing model using various numerical methods.

Let $V(x, y, t)$ be the equity swap derivative whose price depends on the short-rate driving variable x and the log-normal stock price y. Then, the governing parabolic equation can be stated as follows (p. 106, [29]):

$$\begin{aligned} \frac{\partial V}{\partial t} &- \frac{\sigma^r(t)^2}{2}\frac{\partial^2 V}{\partial x^2} - \rho\sigma^r(t)\sigma^S(y,t)\frac{\partial^2 V}{\partial_x \partial_y} - \frac{\sigma^S(y,t)^2}{2}\frac{\partial^2 V}{\partial y^2} \\ &+ \kappa(t)x\frac{\partial V}{\partial x} - \zeta(x,y,t)\frac{\partial V}{\partial y} - r(t)V = 0, \end{aligned} \tag{1.1}$$

where $\zeta(x,y,t) = r(t) - f(0,t) - \frac{\sigma^S(y,t)^2}{2}$. Here, $\sigma^r(t)$, $\sigma^S(y,t)$, ρ, and $\kappa(t)$ designate volatility parameters and $f(t,T)$ denotes the forward rate which is the instantaneous forward interest rate prevailing at time t for the maturity $T > t$. The instantaneous interest rate $r(t)$ fulfills $r(t) = f(t,t)$.

Notice that the coefficients in (1.1) depend on time and space variables in some special fashion. Such an equations can be solved by using standard numerical methods [4,12,39]. However, in real application fields, one has to cover thousands of derivatives in a few hours. In order to resolve this situation, it is necessary to develop a very efficient numerical method to save time. A high-order compact scheme reduces computational costs significantly, and thus one can expect more correct results in almost similar time compared to the standard second-order schemes.

As our aim is to develop a fourth–order compact scheme for a class of parabolic equations which are slightly generalized from (1.1), let us briefly discuss higher–order compact schemes. The key idea of deriving a high-order compact finite difference scheme begins with replacing high-order terms in the Taylor expansion with derivatives of the forcing term [36]. Using this basic idea, many researchers have developed fourth- and higher-order schemes for constant and variable coefficients in two and three dimensions. Usually a high-order compact scheme denotes spatial approximations, and one can easily adopt time-wise approximation using high-order time stepping methods, such as Runge-Kutta fourth-order method and the Richardson method. For parabolic partial differential equations including convection terms with time-dependent coefficients, high-order compact finite difference schemes draw constant attention from researchers since these equations are very closely related to real applications, such as option pricing that we are discussing in this thesis.

The organization of the thesis is as follows. In the chapter to follow, we will introduce a coordinate transformation to remove mixed derivative in the governing parabolic equation. We will the derive a nine-point compact scheme for the resulting equation. In Chapter 3 a stability analysis will be performed and show that the nine-point compact scheme is of fourth order

and unconditionally stable. In Chapter 4 several numerical examples are shown to confirm the validity of the proposed scheme.

1.1 Previous Studies

Let us briefly review previous studies. Spotz and Carey [35] derived a fourth-order compact finite different scheme for the one-dimensional unsteady convection-diffusion equation. Li, Chen, and Liu [25] developed a sixth-order scheme for

$$\frac{\partial u}{\partial t} - \alpha \frac{\partial^2 u}{\partial x^2} + c \frac{\partial u}{\partial x} = f(x),$$

where α and c are constant coefficients. Next, Karaa and Zhang [23] and Tian and Ge [37, 38] expanded Spotz–Carey's study to two dimensional space with the ADI (Alternating Directional Implicit) method. The ADI method is very practical since the resulting matrices from discretization are decomposed direction by direction so that the linear systems are easily solvable. Combining high order schemes with ADI method, Karaa and Zhang [23] suggested a fourth-order compact ADI scheme for the following type of equation:

$$\frac{\partial u}{\partial t} - \alpha_x \frac{\partial^2 u}{\partial x^2} - \alpha_y \frac{\partial^2 u}{\partial y^2} + c_x \frac{\partial u}{\partial x} + c_y \frac{\partial u}{\partial y} = f(x, y), \tag{1.2}$$

where α_x, α_y, c_x, and c_y are constant coefficients. Notice that (1.2) lacks a mixed derivative term. In [20], Karaa suggested a fourth–order discretization scheme with a variable mixed derivative coefficient term. However, the mixed derivative does not allow discretization to be used with the ADI method because the direction will not be fully decomposed separately. Later, Geiser [14] designed a splitting method, which is similar

to the ADI method, to the following equation:

$$\frac{\partial u}{\partial t} - \alpha_x \frac{\partial^2 u}{\partial x^2} - \alpha_y \frac{\partial^2 u}{\partial y^2} + \beta(x,y)\frac{\partial^2 u}{\partial_x \partial_y} + c_1(x,y)\frac{\partial u}{\partial x} + c_2(x,y)\frac{\partial u}{\partial y} = f(x,y), \tag{1.3}$$

where α_x and α_y are constant coefficients and $\beta(x,y)$, $c_1(x,y)$, and $c_2(x,y)$ are sufficiently smooth functions. Again, Karaa [21] studied the ADI method for (1.3) in slightly more general form:

$$\begin{aligned} \frac{\partial u}{\partial t} &- \alpha_x \frac{\partial^2 u}{\partial x^2} - \alpha_y \frac{\partial^2 u}{\partial y^2} + \beta(x,y)\frac{\partial^2 u}{\partial_x \partial_y} \\ &+ c_1(x,y)\frac{\partial u}{\partial x} + c_2(x,y)\frac{\partial u}{\partial y} + d(x,y)u = f(x,y), \end{aligned} \tag{1.4}$$

where the assumptions on coefficients are identical to those in (1.3) and $d(x,y)$ is a sufficiently smooth function. Three dimensional problems have been studied by Karaa [22], Qin [32], and Ma and Ge [26].

Notice that the coefficients in the equity swap derivative equation governed by (1.1) do not fulfill the assumptions in the above developed compact schemes. In particular the mixed derivative term in (1.1) needs some special treatment. We will apply a coordinate transformation to remove it. Then a standard approach to develop fourth-order compact schemes will be applied.

1.2 Equity Swaps

According to [10], "an equity swap is a transaction between two parties in which each party agrees to make a series of payments to the other, with at least one set of payments determined by the return on a stock or stock index. The return is calculated based on a given notional principal and may or may not include dividends. The payments occur on regularly scheduled dates over a specified period of time." For instance, two parties may agree to swap a floating interest rate against some stock index such as FTSE (the share index of the 100 most highly capitalized UK companies

listed on the London Stock Exchange).

In this section we give a brief introduction to interest swaps following the exposition given in [29]. By T and t denote by the maturity and current times. A yield curve is a graphical representation of the relation between the time to maturity $\tau = T - t$ and the annual percent yield that is currently available in the marketplace. It is expressed in terms of either the "continuously-compounded spot interest rate prevailing at time t for the maturity $T > t$" $R(t,T)$ or the "instantaneous forward interest rate prevailing at time t for the maturity $T > t$" (simply "forward rate") $f(t,T)$, fixed at time t for instantaneous borrowing at time T. Then the zero coupon bond price $P(t,T)$ and the forward rate are given by

$$\begin{aligned} P(t,T) &= e^{-R(t,T)\tau}, \\ f(t,T) &= -\frac{\partial}{\partial T}\log P(t,T) = \frac{\partial R}{\partial T}(t,T)\tau + R(t,T). \end{aligned}$$

Typical examples of interest models include $LIBOR$ market and short rate models, which are, respectively, given by

$$\begin{aligned} LIBOR(t,T) &= \frac{1}{\tau}\left[\frac{1}{P(t,T)} - 1\right], \\ r_t &= f(t,t). \end{aligned}$$

$LIBOR(t,T)$ is the simply-compounded spot interest rate prevailing at time t for the maturity T [9].However, interest rate models usually have term structures which depend on segments of forward curves, which are described by stochastic time evolution of the entire forward curve. Among such short rates, one-factor model driven by one stochastic driver, satisfies the following stochastic differential equation:

$$dr_t = \mu(t,r_t)dt + \sigma(t,r_t)dW_t, \tag{1.5}$$

where W_t is a Brown motion in the risk-neutral measure Q, and μ and σ denote the drift and diffusion coefficients. Owing to the Black-Scholes

formula, all assets (say S_t), priced in terms of the money market, are martingales with respect to the risk-neutral measure Q.

The HJM model and its descendants belong to the class of stochastic LIBOR models while Vasicek model and its variants belong to stochastic short rate models. Among stochastic short-rate models, let us look at the Hull-White model described by

$$dr_t = (\theta_t - \kappa_t r_t)dt + \sigma_t dW_t. \tag{1.6}$$

where θ_t denotes the drift while κ_t and σ_t denote the volatility parameters.

Decompose the interest rate r_t given by the Hull-White stochastic differential equation (1.6) as follows:

$$r_t = X_t + \overline{X}_t, \tag{1.7}$$

where $\overline{X}_t$ denotes the deterministic function representing interest rate with term-structure satisfying

$$d\overline{X}_t = (\theta_t - \kappa_t \overline{X}_t)dt; \qquad \overline{X}_0 = f(0,0). \tag{1.8}$$

and X_t satisfies the Ornstein-Uhlenbeck process

$$dX_t = -\kappa_t X_t dt + \sigma_t dW_t; \qquad X_0 = 0. \tag{1.9}$$

Set $\Lambda_{st} = \int_s^t \kappa_u du$. By using integrating factors, the solutions of (1.8) and (1.9) are given by

$$\begin{aligned} \overline{X}_t &= \overline{X}_0 e^{-\Lambda_{0t}} + \int_0^t \theta_u e^{-\Lambda_{ut}}\, du, \\ X_t &= X_s e^{-\Lambda_{st}} + \int_s^t \sigma_u e^{-\Lambda_{ut}} dW_u, \quad \forall 0 \le s \le t, \end{aligned}$$

from which the solution of (1.6) is given by

$$r_t = f(0,0)e^{-\Lambda_{0t}} + \int_0^t \theta_s e^{-\Lambda_{st}}\, ds + \int_0^t \sigma_s e^{-\Lambda_{st}} dW_s. \tag{1.10}$$

Denote

$$\widehat{M}(t,T) = \int_t^T e^{-\Lambda_{ts}}\ ds = \int_t^T e^{-\int_t^s \kappa_u\ du}\ ds.$$

Then the price of a zero-coupon bond $P(t,T)$ is given by the following formula [29]:

$$\frac{P(0,T)}{P(0,t)} \exp\left(-X_t\widehat{M}(t,T) + \frac{1}{2}\int_0^t \sigma_s^2 \left[\widehat{M}(s,t)^2\ - \widehat{M}(s,T)^2\right]\ ds\right) \tag{1.11}$$

Consider the stock price model:

$$dS_t = (r_t - v_t)S_t dt + \sigma^S(S_t,t)S_t dW_t^S, \tag{1.12}$$

where v_t incorporates the dividends and the repo rate, assuming that the dividend is assume to be proportional to the stock price.

Also, consider the interest rates following an Ornstein-Uhlenbeck model as in (1.9)

$$dX_t = -\kappa_t X_t dt + \sigma_t^r dW_t^r, \tag{1.13}$$

with the correlation

$$\left\langle dW_t^r, dW_s^S \right\rangle = \rho dt.$$

In order to avoid the dependency of the dividends and repo, apply the changes of variables from S_t to y_t:

$$S_t = S_0 e^{y_t} \frac{e^{-\int_0^t v_s ds}}{P(0,t)},$$

Then, due to Ito formula and

$$\frac{\frac{\partial p(0,t)}{\partial T}}{P(0,t)} = -\frac{\partial \log P(0,t)}{\partial T} = f(0,t),$$

(1.12) leads to

$$dy_t = \left[r_t - f(0,t) - \frac{1}{2}(\sigma_t^S)^2 \right] dt + \sigma_t^S dW_t. \tag{1.14}$$

Let $V(x,y,t)$ be the derivative whose price depends on the short-rate driving variable (X_t) and the stock price (S_t).

Let $\mathcal{L}_t$ denote the time-dependent second-order elliptic partial differential operator:

$$\mathcal{L}_t = \frac{(\sigma_t^r)^2}{2}\partial_x^2 + \rho\sigma_t^r\sigma_t^S\partial_x\partial_y + \frac{(\sigma_t^S)^2}{2}\partial_y^2 - \kappa_t x\partial_x + \left(r_t - f(0,t) - \frac{1}{2}(\sigma_t^S)^2 \right)\partial_y$$

Recall that

$$\sigma_t^R = \sigma^R(t), \quad \sigma_t^S = \sigma^S(S_t,t)$$

Since the value of the derivative discounted by the money market account is a martingale,

$$d\left(\frac{V}{B}\right) = \left[\partial_t V + \mathcal{L}_t V \ - r_t V\right] dt + \text{a martingale part} \tag{1.15}$$

it follows that V satisfies

$$\partial_t V + \mathcal{L}_t V - r_t V = 0. \tag{1.16}$$

- The forward curve $f(0,t)$ is discontinuous
- Thus, the short rate r_t is discontinuous
- $r_t - f(0,t)$ has smaller discontinuities
- $r_t - f(0,t)$ continuous in the Hull-White model and in the limit of

zero interest rate volatility

Writing $U(x, y, t) = V(x, y, t)P(0, t)$, one has

$$\partial_t U + L_t U - (r_t - f(0, t))U = 0. \tag{1.17}$$

$$r_t - f(0, t) = X_t + \int_0^t \sigma_s^2 \widehat{M}(s, t) e^{-\Lambda_{st}} \, ds \tag{1.18}$$

Chapter 2

Higher–order compact Finite difference scheme

Consider the parabolic problem:

$$\partial_t u - \mathcal{L}u = f(x, y, t), \tag{2.1}$$

where $\mathcal{L} = a_1\partial_x^2 + a_2\partial_y^2 + b\partial_x\partial_y + c_1\partial_x + c_2\partial_y + d$, whose coefficients may depend on x, y, and t partially as follows:

$$\begin{gathered} a_1 = a_1(t), \quad a_2 = a_2(y, t), \quad b = b(y, t), \\ c_1 = c_1(x, y, t), \quad c_2 = c_2(x, y, t), \quad d = d(x, y, t). \end{gathered}$$

Temporarily we write

$$\mathcal{L}u = \partial_t u - f(x, y, t) \equiv F(x, y, t, u(x, y, t)) \tag{2.2}$$

2.1 Seeking a higher-order scheme

By Taylor expansion, we have

$$\begin{aligned}\mathcal{L}u &= a_1\left(\delta_x^2 u - \frac{\Delta x^2}{12}\partial_x^4 u\right) + a_2\left(\delta_y^2 u - \frac{\Delta y^2}{12}\partial_y^4 u\right)\\ &\quad + b\left(\delta_x^0\delta_y^0 u - \frac{\Delta x^2}{6}\partial_x^3\partial_y u - \frac{\Delta y^2}{6}\partial_x\partial_y^3 u\right)\\ &\quad + c_1\left(\delta_x^0 u - \frac{\Delta x^2}{6}\partial_x^3 u\right) + c_2\left(\delta_y^0 u - \frac{\Delta y^2}{6}\partial_y^3 u\right) + du\\ &\quad + \mathcal{O}(\Delta x^4) + \mathcal{O}(\Delta y^4)\end{aligned}$$

since

$$\begin{aligned}&\partial_x^2 u = \delta_x^2 u - \frac{\Delta x^2}{12}\partial_x^4 u + \mathcal{O}(\Delta x^4), \quad \partial_y^2 u = \delta_y^2 u - \frac{\Delta y^2}{12}\partial_y^4 u + \mathcal{O}(\Delta y^4),\\ &\partial_x\partial_y u = \delta_x^0\delta_y^0 u - \frac{\Delta x^2}{6}\partial_x^3\partial_y u - \frac{\Delta y^2}{6}\partial_x\partial_y^3 u + \mathcal{O}(\Delta x^4) + \mathcal{O}(\Delta y^4),\\ &\partial_x u = \delta_x^0 u - \frac{\Delta x^2}{6}\partial_y^3 u + \mathcal{O}(\Delta x^4), \quad \partial_y u = \delta_y^0 u - \frac{\Delta y^2}{6}\partial_y^3 u + \mathcal{O}(\Delta y^4),\end{aligned}$$

where the following standard finite difference approximations in [36] are used:

$$\begin{aligned}\delta_x^0 u_{j,k} &= \frac{u_{j+1,k} - u_{j-1,k}}{2\Delta x}, \quad \delta_y^0 u_{j,k} = \frac{u_{j,k+1} - u_{j,k-1}}{2\Delta y},\\ \delta_x^2 u_{j,k} &= \frac{u_{j+1,k} - 2u_{j,k} + u_{j-1,k}}{\Delta x^2}, \quad \delta_y^2 u_{j,k} = \frac{u_{j,k+1} - 2u_{j,k} + u_{j,k+1}}{\Delta y^2}.\end{aligned}$$

Next, in order to have a nine point compact scheme, we need to replace the terms containing derivatives of order higher than or equal to three that would not be approximated by a nine point finite difference. For this, we differentiate (2.2) to obtain the following:

$$\begin{aligned}\partial_x F &= \Big[a_1\partial_x^3 + a_2\partial_x\partial_y^2 + b\partial_x^2\partial_y + c_1\partial_x^2 + c_2\partial_x\partial_y\Big]u \qquad (2.3)\\ &\quad + \Big[(\partial_x a_1)\partial_x^2 + (\partial_x a_2)\partial_y^2 + (\partial_x b)\partial_x\partial_y + (\partial_x c_1)\partial_x + (\partial_x c_2)\partial_y\Big]u,\end{aligned}$$

$$\begin{aligned}\partial_x^2 F &= \Big[a_1\partial_x^4 + a_2\partial_x^2\partial_y^2 + b\partial_x^3\partial_y + c_1\partial_x^3 + c_2\partial_x^2\partial_y\Big]u \\ &\quad +2\Big[(\partial_x a_1)\partial_x^3 + (\partial_x a_2)\partial_x\partial_y^2 + (\partial_x b)\partial_x^2\partial_y + (\partial_x c_1)\partial_x^2 + (\partial_x c_2)\partial_x\partial_y\Big]u \\ &\quad +\Big[(\partial_x^2 a_1)\partial_x^2 + (\partial_x^2 a_2)\partial_y^2 + (\partial_x^2 b)\partial_x\partial_y + (\partial_x^2 c_1)\partial_x + (\partial_x c_2)\partial_y\Big]u,\end{aligned} \tag{2.4}$$

$$\begin{aligned}\partial_y F &= \Big[a_1\partial_x^2\partial_y + a_2\partial_y^3 + b\partial_x\partial_y^2 + c_1\partial_x\partial_y + c_2\partial_y^2\Big]u \\ &\quad +\Big[(\partial_y a_1)\partial_x^2 + (\partial_y a_2)\partial_y^2 + (\partial_y b)\partial_x\partial_y + (\partial_y c_1)\partial_x + (\partial_y c_2)\partial_y\Big]u,\end{aligned} \tag{2.5}$$

$$\begin{aligned}\partial_y^2 F &= \Big[a_1\partial_x^2\partial_y^2 + a_2\partial_y^4 + b\partial_x\partial_y^3 + c_1\partial_x\partial_y^2 + c_2\partial_y^3\Big]u \\ &\quad +2\Big[(\partial_y a_1)\partial_x^2\partial_y + (\partial_y a_2)\partial_y^3 + (\partial_y b)\partial_x\partial_y^2 + (\partial_y c_1)\partial_x\partial_y + (\partial_y c_2)\partial_y^2\Big]u \\ &\quad +\Big[(\partial_y^2 a_1)\partial_x^2 + (\partial_y^2 a_2)\partial_y^2 + (\partial_y^2 b)\partial_x\partial_y + (\partial_y^2 c_1)\partial_x + (\partial_y^2 c_2)\partial_y\Big]u.\end{aligned} \tag{2.6}$$

The term $\partial_x^3 u$ is approximated by differentiating (2.2) with respect to x. That is,

$$\begin{aligned}\partial_x F &= \Big[a_1\partial_x^3 + a_2\partial_x\partial_y^2 + b\partial_x^2\partial_y + c_1\partial_x^2 + c_2\partial_x\partial_y + d\partial_x\Big]u \\ &\quad + \Big[(\partial_x a_1)\partial_x^2 + (\partial_x a_2)\partial_y^2 + (\partial_x b)\partial_x\partial_y + (\partial_x c_1)\partial_x + (\partial_x c_2)\partial_y + (\partial_x d)\Big]u,\end{aligned}$$

and it yields a nine point compact scheme for $\partial_x^3 u$ with second order accuracy,

$$\begin{aligned}\partial_x^3 u = \frac{1}{a_1}\Big[\partial_x F &- \Big[a_2\delta_x^0\delta_y^2 + b\delta_x^2\delta_y^0 + c_1\delta_x^2 + c_2\delta_x^0\delta_y^0 + d\delta_x^0\Big]u \\ &+ \Big[(\partial_x a_1)\delta_x^2 + (\partial_x a_2)\delta_y^2 + (\partial_x b)\delta_x^0\delta_y^0 + (\partial_x c_1)\delta_x^0 + (\partial_x c_2)\delta_y^0 \\ &\quad + (\partial_x d)\Big]u\Big] + \mathcal{O}(\Delta x^2).\end{aligned}$$

The same technique is applied for approximating $\partial_y^3 u$. If one substitutes the approximations of $\partial_x^3 u$ and $\partial_y^3 u$ to (2.3), it still preserves the fourth order accuracy by the virtue of h_x^2 and h_y^2 terms as multipliers while

keeping the nine point compact stencil.

In order to approximate the other differential operators, $\partial_x^4, \partial_x^3\partial_y, \partial_x\partial_y^3$, and ∂_y^4, we differentiate (2.2) twice with respect to x and y. To treat ∂_x^3 and ∂_y^3, we need to expand the differential operators in second order accurate forms. If we consider the differential operators as variables of a system of algebraic equations, then it is written in a form,

$$a_1A_1 + bA_3 = b_1, \quad a_2A_2 + bA_4 = b_2, \quad a_1A_3 + a_2A_4 = b_3,$$

where

$$A_1 = \partial_x^4 u, \quad A_2 = \partial_y^4 u, \quad A_3 = \partial_x^3\partial_y u, \quad A_4 = \partial_x\partial_y^3 u.$$

Since the system of equations is under-determined, the system is not solvable and thus the approximation with a fourth-order nine-point compact scheme is infeasible through this procedure. Instead, we can think of the following options:

1. Remove the mixed derivative from (2.1).

2. Make a_1 and a_2 be equal.

Option 2 is only applicable in a very restricted situation. Especially, if the coefficients a_1 and a_2 are given as functions, which we are interested in, then option 2 is even more implausible. Note that we consider a problem for which coefficients a_1 and a_2 are only dependent on y and t. In this case, we can remove the mixed derivative term by introducing a special coordinate transformation that is described in the following subsection.

2.2 Coordinate transformation

For the sake of convenience, we denote by (x, y, t) the transformed variables and by $(\hat{x}, \hat{y}, \hat{t})$ the original variables. We specify the relation between the two coordinates in such a way that

$$(x, y, t) = \left(\phi(\hat{x}, \hat{y}, \hat{t}), \hat{y}, \hat{t}\right), \tag{2.7}$$

where ϕ is determined later in this subsection. We distinguish the original and the transformed dependent variables by adding $\widehat{\ }$s to the original ones. For instance,

$$\widehat{u}(\hat{x}, \hat{y}, \hat{t}) = u(x, y, t), \quad \widehat{\sigma^r}(\hat{t}) = \sigma^r(t), \text{ and } \quad \widehat{\sigma^S}(\hat{y}, \hat{t}) = \sigma^S(y, t).$$

Since the purpose is to remove the mixed derivative term, it is enough to consider the second derivative terms from the equation (1.1). We define

$$\mathcal{L}^{(2)}_{\hat{t}}\widehat{u}(\hat{x}, \hat{y}, \hat{t}) = \left[\frac{\widehat{\sigma^r}(\hat{t})^2}{2}\partial^2_{\hat{x}} + \rho\widehat{\sigma^r}(\hat{t})\widehat{\sigma^S}(\hat{y}, \hat{t})\partial_{\hat{x}}\partial_{\hat{y}} + \frac{\widehat{\sigma^S}(\hat{y}, \hat{t})^2}{2}\partial^2_{\hat{y}}\right]\widehat{u}(\hat{x}, \hat{y}, \hat{t}) \tag{2.8}$$

By the chain rule, we immediately have

$$\begin{aligned}
\partial_{\hat{x}}\widehat{u}(\hat{x}, \hat{y}, \hat{t}) &= \partial_x u(x, y, t)\partial_{\hat{x}}\phi(\hat{x}, \hat{y}, \hat{t}), && \text{(2.9a)}\\
\partial_{\hat{y}}\widehat{u}(\hat{x}, \hat{y}, \hat{t}) &= \partial_x u(x, y, t)\partial_{\hat{y}}\phi(\hat{x}, \hat{y}, \hat{t}) + \partial_y u(x, y, t), && \text{(2.9b)}\\
\partial_{\hat{t}}\widehat{u}(\hat{x}, \hat{y}, \hat{t}) &= \partial_x u(x, y, t)\partial_{\hat{t}}\phi(\hat{x}, \hat{y}, \hat{t}) + \partial_t u(x, y, t), && \text{(2.9c)}\\
\partial^2_{\hat{x}}\widehat{u}(\hat{x}, \hat{y}, \hat{t}) &= \partial^2_x u(x, y, t)[\partial_{\hat{x}}\phi(\hat{x}, \hat{y}, \hat{t})]^2 + \partial_x u(x, y, t)\partial^2_{\hat{x}}\phi(\hat{x}, \hat{y}, \hat{t}), && \text{(2.9d)}\\
\partial^2_{\hat{y}}\widehat{u}(\hat{x}, \hat{y}, \hat{t}) &= \partial^2_x u(x, y, t)[\partial_{\hat{y}}\phi(\hat{x}, \hat{y}, \hat{t})]^2 && \text{(2.9e)}\\
&\quad +2\partial_x\partial_y u(x, y, t)\partial_{\hat{y}}\phi(\hat{x}, \hat{y}, \hat{t})\\
&\quad +\partial^2_y u(x, y, t) + \partial_x u(x, y, t)\partial^2_{\hat{y}}\phi(\hat{x}, \hat{y}, \hat{t}),\\
\partial_{\hat{x}}\partial_{\hat{y}}\widehat{u}(\hat{x}, \hat{y}, \hat{t}) &= \partial^2_x u(x, y, t)\partial_{\hat{x}}\phi(\hat{x}, \hat{y}, \hat{t})\partial_{\hat{y}}\phi(\hat{x}, \hat{y}, \hat{t}) && \text{(2.9f)}\\
&\quad +\partial_x\partial_y u(x, y, t)\partial_{\hat{x}}\phi(\hat{x}, \hat{y}, \hat{t}) + \partial_x u(x, y, t)\partial_{\hat{x}}\partial_{\hat{y}}\phi(\hat{x}, \hat{y}, \hat{t}).
\end{aligned}$$

Applying (2.9d)–(2.9f) to (2.8) leads to

$$\begin{aligned}\mathcal{L}_{\hat{t}}^{(2)}\widehat{u}(\hat{x},\hat{y},\hat{t}) \;=\; & \Bigg[\frac{\widehat{\sigma^r}(\hat{t})^2}{2}\left(\partial_{\hat{x}}\phi(\hat{x},\hat{y},\hat{t})\right)^2 \\ & +\rho\widehat{\sigma^r}(\hat{t})\widehat{\sigma^S}(\hat{y},\hat{t})\partial_{\hat{x}}\phi(\hat{x},\hat{y},\hat{t})\partial_{\hat{y}}\phi(\hat{x},\hat{y},\hat{t}) \\ & +\frac{\widehat{\sigma^S}(\hat{y},\hat{t})^2}{2}\left(\partial_{\hat{x}}\phi(\hat{x},\hat{y},\hat{t})\right)^2\Bigg]\partial_x^2 u(x,y,t) \\ & +\frac{\widehat{\sigma^S}(\hat{y},\hat{t})^2}{2}\partial_y^2 u(x,y,t) \\ & +\Big[\rho\widehat{\sigma^r}(\hat{t})\widehat{\sigma^S}(\hat{y},\hat{t})\partial_{\hat{x}}\phi(\hat{x},\hat{y},\hat{t}) \\ & \quad +\widehat{\sigma^S}(\hat{y},\hat{t})^2\partial_{\hat{y}}\phi(\hat{x},\hat{y},\hat{t})\Big]\partial_x\partial_y u(x,y,t) \\ & +\{\text{lower order terms}\}.\end{aligned} \tag{2.10}$$

At this point $\phi(\hat{x},\hat{y},\hat{t})$ should be determined such that the coefficient of the mixed derivative term in (2.10) vanishes. This amounts to solving the first–order (linear) partial differential equation:

$$\rho\widehat{\sigma^r}(\hat{t})\widehat{\sigma^S}(\hat{y},\hat{t})\partial_{\hat{x}}\phi(\hat{x},\hat{y},\hat{t})+\widehat{\sigma^S}(\hat{y},\hat{t})^2\partial_{\hat{y}}\phi(\hat{x},\hat{y},\hat{t})=0,\quad \hat{x}>0,\hat{y}>0.$$

Using the method of characteristics one immediately finds the solution in the form:

$$\phi(\hat{x},\hat{y},\hat{t})=\hat{x}-\rho\widehat{\sigma^r}(\hat{t})\int_0^{\hat{y}}\frac{1}{\widehat{\sigma^S}(\xi,\hat{t})}d\xi. \tag{2.11}$$

The coefficient of $\partial_x^2 u(x,y,t)$ is determined by substituting (2.11) into (2.10), that is,

$$\begin{aligned}&\frac{\sigma^r(\hat{t})^2}{2}\left(\partial_{\hat{x}}\phi(\hat{x},\hat{y},\hat{t})\right)^2+\rho\sigma^r(\hat{t})\sigma^S(\hat{y},\hat{t})\partial_{\hat{x}}\phi(\hat{x},\hat{y},\hat{t})\partial_{\hat{y}}\phi(\hat{x},\hat{y},\hat{t}) \\ &+\frac{\sigma^S(\hat{y},\hat{t})^2}{2}\left(\partial_{\hat{y}}\phi(\hat{x},\hat{y},\hat{t})\right)^2=\frac{\sigma^r(\hat{t})^2}{2}\left(1-\rho^2\right).\end{aligned}$$

This implies that the coefficients of $\partial_x^2 u(x,y,t)$ and $\partial_y^2 u(x,y,t)$ are

given by $(\frac{1}{2} - \frac{1}{2}\rho^2)\widehat{\sigma^r}(\hat{t})^2$ and $\frac{1}{2}\widehat{\sigma^S}(\hat{y},\hat{t})^2$, respectively. Thus we conclude under the assumption that

$$\left(\frac{1}{2} - \frac{1}{2}\rho^2\right)\widehat{\sigma^r}(\hat{t})^2 > 0, \tag{2.12}$$

the coordinate transformation (2.7) with ϕ given by (2.11) makes the mixed derivative term vanish with preserving the ellipticity of $\mathcal{L}^{(2)}_{\hat{t}}$.

Remark 2.1. The assumption (2.12) holds if the correlation ρ between the Brownian motions W^r and W^S is less than 1 and the volatility of W^S does not vanish for all $t > 0$.

Finally, by expanding the lower order terms we have a transformed equation,

$$\begin{aligned}
\mathcal{L}_{\hat{t}}\widehat{u}(\hat{x},\hat{y},\hat{t}) =& \frac{\sigma^r(t)^2}{2}\left(1-\rho^2\right)\partial_x^2 u(x,y,t) + \frac{\sigma^S(y,t)^2}{2}\partial_y^2 u(x,y,t) \\
&+ \left[\frac{\rho\sigma^r(t)\left(\partial_{\hat{y}}\sigma^S(y,t)\right)}{2} - \kappa_t x - \zeta(x,y,t)\frac{\rho\sigma^r(t)}{\sigma^S(y,t)}\right. \\
&\quad + \rho\partial_t\left(\sigma^r(t)\right)\int_0^y \frac{1}{\sigma^S(\xi,t)}d\xi \\
&\quad \left. + \rho\sigma^r(t)\partial_t\left(\int_0^y \frac{1}{\sigma^S(\xi,t)}d\xi\right)\right]\partial_x u(x,y,t) \\
&+ \zeta(x,y,t)\partial_y u(x,y,t) - \gamma(x,y,t)u(x,y,t).
\end{aligned} \tag{2.13}$$

2.3 A nine-point compact scheme

Since the resulting PDE does not contain a mixed derivative, now we return to the approximation of (2.2) with $b = 0$. The third derivatives ∂_x^3 and ∂_y^3 are already approximated in §2.1. The second derivative of F with respect to x yields an approximation of ∂_x^4 with second-order accuracy. Since

$$
\begin{aligned}
\partial_x F &= \Big[a_1\partial_x^3 + a_2\partial_x\partial_y^2 + c_1\partial_x^2 + c_2\partial_x\partial_y + d\partial_x\Big]u \\
&\quad + \Big[(\partial_x a_1)\partial_x^2 + (\partial_x a_2)\partial_y^2 + (\partial_x c_1)\partial_x + (\partial_x c_2)\partial_y + (\partial_x d)\Big]u, \\
\partial_x^2 F &= \Big[a_1\partial_x^4 + a_2\partial_x^2\partial_y^2 + c_1\partial_x^3 + c_2\partial_x^2\partial_y + d\partial_x^2\Big]u \\
&\quad + 2\Big[(\partial_x a_1)\partial_x^3 + (\partial_x a_2)\partial_x\partial_y^2 + (\partial_x c_1)\partial_x^2 + (\partial_x c_2)\partial_x\partial_y + (\partial_x d)\partial_x\Big]u \\
&\quad + \Big[(\partial_x^2 a_1)\partial_x^2 + (\partial_x^2 a_2)\partial_y^2 + (\partial_x^2 c_1)\partial_x + (\partial_x^2 c_2)\partial_y + (\partial_x^2 d)\Big]u, \\
\partial_y F &= \Big[a_1\partial_x^2\partial_y + a_2\partial_y^3 + c_1\partial_x\partial_y + c_2\partial_y^2 + d\partial_y\Big]u \\
&\quad + \Big[(\partial_y a_1)\partial_x^2 + (\partial_y a_2)\partial_y^2 + (\partial_y c_1)\partial_x + (\partial_y c_2)\partial_y + (\partial_y d)\Big]u, \\
\partial_y^2 F &= \Big[a_1\partial_x^2\partial_y^2 + a_2\partial_y^4 + c_1\partial_x\partial_y^2 + c_2\partial_y^3 + d\partial_y^2\Big]u \\
&\quad + 2\Big[(\partial_y a_1)\partial_x^2\partial_y + (\partial_y a_2)\partial_y^3 + (\partial_y c_1)\partial_x\partial_y + (\partial_y c_2)\partial_y^2 + (\partial_y d)\partial_y\Big]u \\
&\quad + \Big[(\partial_y^2 a_1)\partial_x^2 + (\partial_y^2 a_2)\partial_y^2 + (\partial_y^2 c_1)\partial_x + (\partial_y^2 c_2)\partial_y + (\partial_y^2 d)\Big]u,
\end{aligned}
$$

one can obtain a second-order approximation of ∂_x^4 in conjunction with the approximation of ∂_x^3. By similar way, ∂_y^4 is approximated with second-order accuracy.

Aggregating all the terms together into (2.2), it becomes

$$
\begin{aligned}
\mathcal{L}u = &\Big[\frac{\Delta x^2}{12}a_2 + \frac{\Delta y^2}{12}a_1\Big]\delta_x^2\delta_y^2 u \\
&+ \Big[\frac{\Delta x^2}{12}c_2 + 2\frac{\Delta y^2}{12}(\partial_y a_1) + \frac{\Delta y^2}{12a_2}(c_2 - 2(\partial_y a_2))a_1\Big]\delta_x^2\delta_y^0 u \\
&+ \Big[\frac{\Delta y^2}{12}c_1 + 2\frac{\Delta x^2}{12}(\partial_x a_2) + \frac{\Delta x^2}{12a_1}(c_1 - 2(\partial_x a_1))a_2\Big]\delta_x^0\delta_y^2 u \\
&+ \Big[2\frac{\Delta x^2}{12}(\partial_x c_2) + 2\frac{\Delta y^2}{12}(\partial_y c_1) \\
&\quad + \frac{\Delta x^2}{12a_1}(c_1 - 2(\partial_x a_1))c_2 + \frac{\Delta y^2}{12a_2}(c_2 - 2(\partial_y a_2))c_1\Big]\delta_x^0\delta_y^0 u \\
&+ \Big[a_1 + \frac{\Delta x^2}{12}(d + 2(\partial_x c_1) + (\partial_x^2 a_1)) + \frac{\Delta y^2}{12}(\partial_y^2 a_1)
\end{aligned}
\tag{2.14}
$$

$$
\begin{aligned}
&\quad + \frac{\Delta x^2}{12a_1}(c_1 - 2(\partial_x a_1))(c_1 + (\partial_x a_1)) + \frac{\Delta y^2}{12a_2}(c_2 - 2(\partial_y a_2))(\partial_y a_1)\Big]\delta_x^2 u \\
&+ \Big[a_2 + \frac{\Delta y^2}{12}(d + 2(\partial_y c_2) + (\partial_y^2 a_2)) + \frac{\Delta x^2}{12}(\partial_x^2 a_2) \\
&\quad + \frac{\Delta y^2}{12a_2}(c_2 - 2(\partial_y a_2))(c_2 + (\partial_y a_2)) + \frac{\Delta x^2}{12a_1}(c_1 - 2(\partial_x a_1))(\partial_x a_2)\Big]\delta_y^2 u \\
&+ \Big[c_1 + \frac{\Delta x^2}{12}(2(\partial_x d) + (\partial_x^2 c_1)) + \frac{\Delta y^2}{12}(\partial_y^2 c_1) \\
&\quad + \frac{\Delta x^2}{12a_1}(c_1 - 2(\partial_x a_1))(d + (\partial_x c_1)) + \frac{\Delta y^2}{12a_2}(c_2 - 2(\partial_y a_2))(\partial_y c_1)\Big]\delta_x^0 u \\
&+ \Big[c_2 + \frac{\Delta y^2}{12}(2(\partial_y d) + (\partial_y^2 c_2)) + \frac{\Delta x^2}{12}(\partial_x^2 c_2) \\
&\quad + \frac{\Delta y^2}{12a_2}(c_2 - 2(\partial_y a_2))(d + (\partial_y c_2)) + \frac{\Delta x^2}{12a_1}(c_1 - 2(\partial_x a_1))(\partial_x c_2)\Big]\delta_y^0 u \\
&+ \Big[d + \frac{\Delta x^2}{12}(\partial_x^2 d) + \frac{\Delta y^2}{12}(\partial_y^2 d) \\
&\quad + \frac{\Delta x^2}{12a_1}(c_1 - 2(\partial_x a_1))(\partial_x d) + \frac{\Delta y^2}{12a_2}(c_2 - 2(\partial_y a_2))(\partial_y d)\Big]u \\
&- F - \frac{\Delta x^2}{12}\delta_x^2 F - \frac{\Delta y^2}{12}\delta_y^2 F \\
&- \frac{\Delta x^2}{12a_1}\Big(c_1 - 2(\partial_x a_1)\Big)\delta_x^0 F - \frac{\Delta y^2}{12a_2}\Big(c_2 - 2(\partial_y a_2)\Big)\delta_y^0 F \\
&+ \mathcal{O}(\Delta x^4) + \mathcal{O}(\Delta y^4).
\end{aligned}
$$

Simply, (2.14) is represented as

$$
\begin{aligned}
\mathcal{L}u \;=\;& L_1\delta_x^2\delta_y^2 u + L_2\delta_x^2\delta_y^0 u + L_3\delta_x^0\delta_y^2 u + L_4\delta_x^0\delta_y^0 u \\
&+ L_5\delta_x^2 u + L_6\delta_y^2 u + L_7\delta_x^0 u + L_8\delta_y^0 u + L_9 u \\
&- A_1\delta_x^2 F - A_2\delta_x^2 F - A_3\delta_x^0 F - A_4\delta_y^0 F + \mathcal{O}(\Delta x^4) + \mathcal{O}(\Delta y^4).
\end{aligned}
$$

By expanding the discretization operators at a point (j,k), we get

$$\begin{aligned}
\mathcal{L}u_{j,k} &= \Big[L_1 - L_2 - L_3 + L_4\Big]u_{j-1,k-1} \\
&+\Big[-2L_1 + 2L_2 + L_6 - L_8\Big]u_{j,k-1} \\
&+\Big[L_1 - L_2 + L_3 - L_4\Big]u_{j+1,k-1} \\
&+\Big[-2L_1 + 2L_3 + L_5 - L_7\Big]u_{j-1,k} \\
&+\Big[4L_1 - 2L_5 - 2L_6 + L_9\Big]u_{j,k} \\
&+\Big[-2L_1 - 2L_3 + L_5 + L_7\Big]u_{j+1,k} \\
&+\Big[L_1 + L_2 - L_3 - L_4\Big]u_{j-1,k+1} \\
&+\Big[-2L_1 - 2L_2 + L_6 + L_8\Big]u_{j,k+1} \\
&+\Big[L_1 + L_2 + L_3 + L_4\Big]u_{j+1,k+1} \\
&-\Big[A_2 - A_4\Big]F_{j,k-1} \\
&-\Big[A_1 - A_3\Big]F_{j-1,k} \\
&-\Big[-2A_1 - 2A_2\Big]F_{j,k} \\
&-\Big[A_1 + A_3\Big]F_{j+1,k} \\
&-\Big[A_2 + A_4\Big]F_{j,k+1} + \mathcal{O}(\Delta x^4) + \mathcal{O}(\Delta y^4).
\end{aligned}$$

Rewrite the equation in a discrete form:

$$\tilde{\mathcal{L}}u = \tilde{\mathcal{A}}F + \mathcal{O}(\Delta x^4) + \mathcal{O}(\Delta y^4),$$

where

$$\tilde{\mathcal{L}} = \begin{pmatrix} L_1 + L_2 - L_3 - L_4 & -2L_1 - 2L_2 + L_6 + L_8 & L_1 + L_2 + L_3 + L_4 \\ -2L_1 + 2L_3 + L_5 - L_7 & 4L_1 - 2L_5 - 2L_6 + L_9 & -2L_1 - 2L_3 + L_5 + L_7 \\ L_1 - L_2 - L_3 + L_4 & -2L_1 + 2L_2 + L_6 - L_8 & L_1 - L_2 + L_3 - L_4 \end{pmatrix},$$

$$\tilde{\mathcal{A}} = \begin{pmatrix} & A_2 + A_4 & \\ A_1 - A_3 & 1 - 2A_1 - 2A_2 & A_1 + A_3 \\ & A_2 - A_4 & \end{pmatrix}.$$

We replace F with $\partial_t u - f$ and apply Crank-Nicolson method in the time axis, then we obtain the final discretization form,

$$\begin{aligned} (\tilde{\mathcal{A}}^{n+1/2} - \frac{\Delta t}{2}\tilde{\mathcal{L}}^{n+1/2})u^{n+1} &= (\tilde{\mathcal{A}}^{n+1/2} + \frac{\Delta t}{2}\tilde{\mathcal{L}}^{n+1/2})u^n \\ &+\Delta t\tilde{\mathcal{A}}^{n+1/2}f^{n+1/2} \\ &+\mathcal{O}(\Delta t^3) + \mathcal{O}(\Delta t \Delta x^4) + \mathcal{O}(\Delta t \Delta y^4). \end{aligned} \tag{2.15}$$

Chapter 3

Stability analysis

In this section, we provide stability conditions for the proposed scheme. First of all, we define discrete Fourier series and its inversion such that

$$\begin{aligned} \hat{u}(\xi) &= \frac{1}{\sqrt{2\pi}} \sum_{m=-\infty}^{\infty} e^{-im\xi} u_m, \\ u_m &= \frac{1}{\sqrt{2\pi}} \int_{-\pi}^{\pi} e^{im\xi} \hat{u}(\xi) d\xi, \end{aligned}$$

where $\xi \in [-\pi, \pi]$ and u_m denotes discrete data points.

The von Neumann analysis simplifies procedures in Fourier analysis as following relationship,

$$u_j^n = g(\xi)^n e^{ij\xi}, \tag{3.1}$$

where i is pure imaginary number $\sqrt{-1}$, j denotes discrete points of the solution, $\xi \in [-\pi, \pi]$ is a frequency coming from discrete Fourier series, and function $g(\xi)$ is an amplification factor that is important for stability.

Also, we can expand (3.1) as two-dimensional case with different phase angle ξ and η:

$$u_{j,k}^n = g(\xi, \eta)^n e^{ij\xi} e^{ik\eta}. \tag{3.2}$$

Our final goal is for $|g(\xi, \eta)^{n+1}/g(\xi, \eta)^n|$ to be bounded by one to guarantee stability. However, the von Neumann analysis is only available for constant

coefficients, so we assume that our scheme has constant coefficient.

Rewrite target equation as follows and freeze all the coefficients so that a_1, a_2, c_1, and c_2 are constant:

$$\partial_t u - (a_1\partial_x^2 + a_2\partial_y^2 + c_1\partial_x + c_2\partial_y + d)u = f$$

In order to simplify the procedure, we adopt the transformation $u = e^{dt}v$ to remove plane term du.

$$(a_1\partial_x^2 + a_2\partial_y^2 + c_1\partial_x + c_2\partial_y)v = \partial_t v - e^{-dt}f \equiv F$$

Therefore, (2.14) will be

$$\begin{aligned}
\mathcal{L}_f v &= \Big[\frac{\Delta x^2}{12}a_2 + \frac{\Delta y^2}{12}a_1\Big]\delta_x^2\delta_y^2 v + \Big[\frac{\Delta x^2}{12}\frac{c_1}{a_1}c_2 + \frac{\Delta y^2}{12}\frac{c_2}{a_2}c_1\Big]\delta_x^0\delta_y^0 v \qquad (3.3)\\
&+\Big[\frac{\Delta x^2}{12}c_2 + \frac{\Delta y^2}{12}\frac{c_2}{a_2}a_1\Big]\delta_x^2\delta_y^0 v + \Big[\frac{\Delta y^2}{12}c_1 + \frac{\Delta x^2}{12}\frac{c_1}{a_1}a_2\Big]\delta_x^0\delta_y^2 v\\
&+\Big[a_1 + \frac{\Delta x^2}{12}\frac{c_1}{a_1}c_1\Big]\delta_x^2 v + \Big[a_2 + \frac{\Delta y^2}{12}\frac{c_2}{a_2}c_2\Big]\delta_y^2 v + c_1\delta_x^0 v + c_2\delta_y^0 v\\
&-F - \frac{\Delta x^2}{12}\delta_x^2 F - \frac{\Delta y^2}{12}\delta_y^2 F\\
&-\frac{\Delta x^2}{12}\frac{c_1}{a_1}\delta_x^0 F - \frac{\Delta y^2}{12}\frac{c_2}{a_2}\delta_y^0 F + \mathcal{O}(\Delta x^4) + \mathcal{O}(\Delta y^4).
\end{aligned}$$

In order to simplify stability analysis, the following operators are introduced in [23]:

$$\begin{aligned}
A_x &= 1 + \frac{\Delta x^2}{12}\delta_x^2 + \frac{\Delta x^2}{12}\frac{c_1}{a_1}\delta_x^0, \quad A_y = 1 + \frac{\Delta y^2}{12}\delta_y^2 + \frac{\Delta y^2}{12}\frac{c_2}{a_2}\delta_y^0,\\
L_x &= \Big[a_1 + \frac{\Delta x^2}{12}\frac{c_1}{a_1}c_1\Big]\delta_x^2 + c_1\delta_x^0, \quad L_y = \Big[a_2 + \frac{\Delta y^2}{12}\frac{c_2}{a_2}c_2\Big]\delta_y^2 + c_2\delta_y^0.
\end{aligned}$$

By simple calculations, we get

$$\begin{aligned}
A_xL_y &= \Big[a_2 + \frac{\Delta y^2}{12}\frac{c_2}{a_2}c_2\Big]\delta_y^2 + c_2\delta_y^0 + \Big[\frac{\Delta x^2}{12}a_2 + \frac{\Delta x^2\Delta y^2}{144}\frac{c_2}{a_2}c_2\Big]\delta_x^2\delta_y^2\\
&+\frac{\Delta x^2}{12}c_2\delta_x^2\delta_y^0 + \Big[\frac{\Delta x^2}{12}\frac{c_1}{a_1}a_2 + \frac{\Delta x^2\Delta y^2}{144}\frac{c_1c_2}{a_1a_2}c_2\Big]\delta_x^0\delta_y^2 + \frac{\Delta x^2}{12}\frac{c_1}{a_1}c_2\delta_x^0\delta_y^0\\
&= \Big[a_2 + \frac{\Delta y^2}{12}\frac{c_2}{a_2}c_2\Big]\delta_y^2 + c_2\delta_y^0 + \frac{\Delta x^2}{12}a_2\delta_x^2\delta_y^2 + \frac{\Delta x^2}{12}c_2\delta_x^2\delta_y^0
\end{aligned}$$

$$+\frac{\Delta x^2}{12}\frac{c_1}{a_1}a_2\delta_x^0\delta_y^2+\frac{\Delta x^2}{12}\frac{c_1}{a_1}c_2\delta_x^0\delta_y^0+\mathcal{O}(\Delta x^2\Delta y^2)$$

$$\begin{aligned}
A_yL_x &= \Big[a_1+\frac{\Delta x^2}{12}\frac{c_1}{a_1}c_1\Big]\delta_x^2+c_1\delta_x^0+\Big[\frac{\Delta y^2}{12}a_1+\frac{\Delta x^2\Delta y^2}{144}\frac{c_1}{a_1}c_1\Big]\delta_x^2\delta_y^2\\
&\quad+\frac{\Delta y^2}{12}c_1\delta_x^0\delta_y^2+\Big[\frac{\Delta y^2}{12}\frac{c_2}{a_2}a_1+\frac{\Delta x^2\Delta y^2}{144}\frac{c_1c_2}{a_1a_2}c_1\Big]\delta_x^2\delta_y^0+\frac{\Delta y^2}{12}\frac{c_2}{a_2}c_1\delta_x^0\delta_y^0\\
&= \Big[a_1+\frac{\Delta x^2}{12}\frac{c_1}{a_1}c_1\Big]\delta_x^2+c_1\delta_x^0+\frac{\Delta y^2}{12}a_1\delta_x^2\delta_y^2+\frac{\Delta y^2}{12}c_1\delta_x^0\delta_y^2\\
&\quad+\frac{\Delta y^2}{12}\frac{c_2}{a_2}a_1\delta_x^2\delta_y^0+\frac{\Delta y^2}{12}\frac{c_2}{a_2}c_1\delta_x^0\delta_y^0+\mathcal{O}(\Delta x^2\Delta y^2)
\end{aligned}$$

$$\begin{aligned}
A_xA_y &= 1+\frac{\Delta x^2}{12}\delta_x^2+\frac{\Delta x^2}{12}\frac{c_1}{a_1}\delta_x^0+\frac{\Delta y^2}{12}\delta_y^2+\frac{\Delta y^2}{12}\frac{c_2}{a_2}\delta_y^0\\
&\quad+\frac{\Delta x^2\Delta y^2}{144}\delta_x^2\delta_y^2+\frac{\Delta x^2\Delta y^2}{144}\frac{c_2}{a_2}\delta_x^2\delta_y^0+\frac{\Delta x^2\Delta y^2}{144}\frac{c_1}{a_1}\delta_x^0\delta_y^2\\
&\quad+\frac{\Delta x^2\Delta y^2}{144}\frac{c_1c_2}{a_1a_2}\delta_x^0\delta_y^0\\
&= 1+\frac{\Delta x^2}{12}\delta_x^2+\frac{\Delta x^2}{12}\frac{c_1}{a_1}\delta_x^0+\frac{\Delta y^2}{12}\delta_y^2+\frac{\Delta y^2}{12}\frac{c_2}{a_2}\delta_y^0+\mathcal{O}(\Delta x^2\Delta y^2)
\end{aligned}$$

With these notations, (3.3) becomes

$$\mathcal{L}_f v=\Big(A_xL_y+A_yL_x\Big)v-A_xA_yF+\mathcal{O}(\Delta x^4)+\mathcal{O}(\Delta y^4). \tag{3.4}$$

Rewrite (3.4) and assume that the forcing term f in F is zero:

$$A_xA_yv_t-(A_xL_y+A_yL_x)v=\mathcal{O}(\Delta x^4)+\mathcal{O}(\Delta y^4). \tag{3.5}$$

Next, apply Crank–Nicolson time discretization to (3.5).

$$\begin{aligned}
\left(A_xA_y-\frac{\Delta t}{2}(A_xL_y+A_yL_x)\right)v^{n+1}=\left(A_xA_y+\frac{\Delta t}{2}(A_xL_y+A_yL_x)\right)v^n \\
+\mathcal{O}(\Delta x^4)+\mathcal{O}(\Delta y^4).
\end{aligned} \tag{3.6}$$

As noticed in [23], the additional term $\frac{\Delta t^2}{4}(v^{n+1}-v^n)$ leads the truncation order

by $\mathcal{O}(\Delta t^3\Delta^2)$, where $\Delta^2 = \Delta x^2 + \Delta y^2$. Then (3.6) becomes

$$\left(A_x - \frac{\Delta t}{2}L_x\right)\left(A_y - \frac{\Delta t}{2}L_y\right)v^{n+1} = \left(A_x + \frac{\Delta t}{2}L_x\right)\left(A_y + \frac{\Delta t}{2}L_y\right)v^n \quad (3.7)$$
$$+ \mathcal{O}(\Delta t^3\Delta^2) + \mathcal{O}(\Delta x^4) + \mathcal{O}(\Delta y^4).$$

Now, plug (3.2) into (3.7) to get amplification factors for each direction. Set G_x^n and G_y^n such that

$$\begin{aligned} G_x^{n+1} &= \left(\left[\frac{5}{6} + \frac{\Delta t}{\Delta x^2}a_1 + \frac{\Delta t}{12}\frac{c_1^2}{a_1}\right] + \right. \\ &\quad \left[\frac{1}{12} + \frac{\Delta x}{24}\frac{c_1}{a_1} - \Delta t\left(\frac{a_1}{2\Delta x^2} + \frac{1}{24}\frac{c_1^2}{a_1} + \frac{\Delta t}{4\Delta x}c_1\right)\right]e^{i\xi} + \\ &\quad \left.\left[\frac{1}{12} - \frac{\Delta x}{24}\frac{c_1}{a_1} - \Delta t\left(\frac{a_1}{2\Delta x^2} + \frac{1}{24}\frac{c_1^2}{a_1} - \frac{\Delta t}{4\Delta x}c_1\right)\right]e^{-i\xi}\right) \\ &= \left(\frac{5}{6} + \frac{1}{6}\cos(\xi)\right) + \Delta t\left(\frac{a_1}{\Delta x^2} + \frac{1}{12}\frac{c_1^2}{a_1}\right)(1 - \cos(\xi)) + \\ &\quad \left(\frac{\Delta x}{12}\frac{c_1}{a_1}\right)i\sin(\xi) - \Delta t\left(\frac{1}{2\Delta x}c_1\right)i\sin(\xi). \end{aligned}$$

Similarly,

$$\begin{aligned} G_y^{n+1} &= \left(\frac{5}{6} + \frac{1}{6}\cos(\eta)\right) + \Delta t\left(\frac{a_2}{\Delta y^2} + \frac{1}{12}\frac{c_2^2}{a_2}\right)(1 - \cos(\eta)) + \\ &\quad \left(\frac{\Delta y}{12}\frac{c_2}{a_2}\right)i\sin(\eta) - \Delta t\left(\frac{1}{2\Delta y}c_2\right)i\sin(\eta), \end{aligned}$$

$$\begin{aligned} G_x^n &= \left(\frac{5}{6} + \frac{1}{6}\cos(\xi)\right) - \Delta t\left(\frac{a_1}{\Delta x^2} + \frac{1}{12}\frac{c_1^2}{a_1}\right)(1 - \cos(\xi)) + \\ &\quad \left(\frac{\Delta x}{12}\frac{c_1}{a_1}\right)i\sin(\xi) + \Delta t\left(\frac{1}{2\Delta x}c_1\right)i\sin(\xi), \end{aligned}$$

$$\begin{aligned} G_y^n &= \left(\frac{5}{6} + \frac{1}{6}\cos(\eta)\right) - \Delta t\left(\frac{a_2}{\Delta y^2} + \frac{1}{12}\frac{c_2^2}{a_2}\right)(1 - \cos(\eta)) + \\ &\quad \left(\frac{\Delta y}{12}\frac{c_2}{a_2}\right)i\sin(\eta) + \Delta t\left(\frac{1}{2\Delta y}c_2\right)i\sin(\eta). \end{aligned}$$

Then (3.7) becomes

$$g^{n+1}G_x^{n+1}G_y^{n+1} = g^n G_x^n G_y^n.$$

Theorem 3.1. *The fourth-order scheme* (2.15) *is unconditionally stable.*

Proof. At first, we prove that

$$\left|\frac{G_x^n}{G_x^{n+1}}\right| \le 1 \text{ and } \left|\frac{G_y^n}{G_y^{n+1}}\right| \le 1.$$

We can use abbreviation for amplification factors such that

$$\begin{aligned}
G_x^{n+1} &= (\alpha_{x,1}+\alpha_{x,2}) + (\alpha_{x,3}-\alpha_{x,4})i, \quad G_x^n = (\alpha_{x,1}-\alpha_{x,2}) + (\alpha_{x,3}+\alpha_{x,4})i,\\
G_y^{n+1} &= (\alpha_{y,1}+\alpha_{y,2}) + (\alpha_{y,3}-\alpha_{y,4})i, \quad G_y^n = (\alpha_{y,1}-\alpha_{y,2}) + (\alpha_{y,3}+\alpha_{y,4})i.
\end{aligned}$$

where

$$\begin{aligned}
\alpha_{x,1} &= \frac{5}{6} + \frac{1}{6}\cos(\xi), \quad \alpha_{y,1} = \frac{5}{6} + \frac{1}{6}\cos(\eta),\\
\alpha_{x,2} &= \Delta t\left(\frac{a_1}{\Delta x^2} + \frac{1}{12}\frac{c_1^2}{a_1}\right)(1-\cos(\xi)), \quad \alpha_{y,2} = \Delta t\left(\frac{a_2}{\Delta y^2} + \frac{1}{12}\frac{c_2^2}{a_2}\right)(1-\cos(\eta)),\\
\alpha_{x,3} &= \frac{\Delta x}{12}\frac{c_1}{a_1}\sin(\xi), \quad \alpha_{y,3} = \frac{\Delta y}{12}\frac{c_2}{a_2}\sin(\eta),\\
\alpha_{x,4} &= \frac{\Delta t}{2\Delta x}c_1\sin(\xi), \quad \alpha_{y,4} = \frac{\Delta t}{2\Delta y}c_2\sin(\eta).
\end{aligned}$$

Notice that the expressions for $|G_x^n/G_x^{n+1}|$ and $|G_y^n/G_y^{n+1}|$ are quite similar, but differ in coefficients and mesh sizes. Let us first look at the ratio in the x-direction:

$$\left|\frac{G_x^n}{G_x^{n+1}}\right| = \frac{|(\alpha_{x,1}-\alpha_{x,2}) + (\alpha_{x,3}+\alpha_{x,4})i|}{|(\alpha_{x,1}+\alpha_{x,2}) + (\alpha_{x,3}-\alpha_{x,4})i|} \le 1 \iff \alpha_{x,1}\alpha_{x,2} \ge \alpha_{x,3}\alpha_{x,4}.$$

Simple trigonometric identities imply that

$$\begin{aligned}
\alpha_{x,1}\alpha_{x,2} &= \frac{\Delta t}{6}\frac{c_1^2}{a_1}\left(1-\frac{1}{3}\sin^2\frac{\xi}{2}\right)\left(\frac{12a_1^2}{\Delta x^2 c_1^2}+1\right)\sin^2\frac{\xi}{2}\\
&\ge \frac{\Delta t}{6}\frac{c_1^2}{a_1}\left(1-\frac{1}{3}\sin^2\frac{\xi}{2}\right)\sin^2\frac{\xi}{2},\\
\alpha_{x,3}\alpha_{x,4} &= \frac{\Delta t}{6}\frac{c_1^2}{a_1}\left(1-\sin^2\frac{\xi}{2}\right)\sin^2\frac{\xi}{2},
\end{aligned}$$

from which it follows that we can prove $\alpha_{x,1}\alpha_{x,2} \geq \alpha_{x,3}\alpha_{x,4}$. Similar arguments for the y-direction hold.

Therefore the unconditionally stable condition is satisfied as follows:

$$\left|\frac{g^{n+1}}{g^n}\right| = \left|\frac{G_x^n G_y^n}{G_x^{n+1} G_y^{n+1}}\right| = \left|\frac{G_x^n}{G_x^{n+1}}\right| \left|\frac{G_y^n}{G_y^{n+1}}\right| \leq 1.$$

This proves the theorem. □

Chapter 4

Numerical results

Example 4.1. The proposed scheme is applied to a problem

$$\partial_t u - \big(a_1\partial_x^2 + a_2\partial_y^2 + c_1\partial_x + c_2\partial_y + d\big)u = f,$$

where

$$\begin{aligned}
a_1(x,t) &= 1+\alpha t\sin\pi x, \quad a_2(y,t) = 1+\beta t\sin\pi y,\\
c_1(x,y,t) &= \xi t(x^2+y^2), \quad c_2(x,y,t) = \eta t(x^2+y^2), \quad d(x,y,t) = \xi\eta txy,\\
f(x,y,t) &= \big[(a_1+a_2-2)\pi^2 - d\big]\, e^{-2\pi^2(t-1.0)}\sin\pi x\sin\pi y\\
&\quad -\pi c_1 e^{-2\pi^2(t-1.0)}\cos\pi x\sin\pi y - \pi c_2 e^{-2\pi^2(t-1.0)}\sin\pi x cos\pi y,
\end{aligned}$$

with computational domain $(x,y,t)\in(0,1)^2\times(0,1]$ This is the form provided by the coordinate transformation that is used for eliminating the mixed derivative term. The exact solution is given in the form,

$$u_{ex}(x,y,t) = e^{-2\pi^2(t-1.0)}\sin\pi x\sin\pi y.$$

Initial data $u(x,y,0)$ and boundary conditions are given by the exact solution. The order of convergence is computed by comparison with the exact solution, where the error is computed in l^∞-norm.

In Table 4.1, N_x, N_y, and N_t denote the number of grid points in the x-axis, y-axis, and time axis, respectively. In order to balance the order of convergence between time and spatial variables, we increase N_t 4-times when increasing 2-

times for N_x and N_y. Table 4.1 shows the order of convergence when $\alpha = 2.0$, $\beta = 5.0$, $\xi = 0.15$, and $\eta = 0.3$. The fourth order convergence is observed as we expect.

(N_x, N_y)	N_t	l^∞-error	Reduction rate
(16, 16)	1024	0.629E-04	
(32, 32)	4096	0.385E-05	4.03
(64, 64)	16384	0.242E-06	3.99

Table 4.1: Convergence result of Example A.2

Example 4.2. The proposed fourth-order nine-point compact scheme is applied for solving the two-dimensional Black-Scholes equation,

$$\partial_t v + \left[-\frac{\sigma_1^2}{2} S_1^2 \partial_{S_1}^2 - \rho\sigma_1\sigma_2 S_1 S_2 \partial_{S_1}\partial_{S_2} - \frac{\sigma_2^2}{2} S_2^2 \partial_{S_2}^2 - rS_1\partial_{S_1} - rS_2\partial_{S_2} \right] v \quad (4.1)$$
$$+ rv = 0$$

where S_i and σ_i denote price and volatility of a stock i for $i = 1, 2$, and v is an option value depending on the basis assets. ρ and r stand for the covariance of the assets and the risk free interest rate, respectively. The original domain $(S_1, S_2, t) \in [0, \infty)^2 \times (0, T]$ is truncated into a finite domain for computational purpose. Initial data is taken from the Cash-or-Nothing option, that is,

$$v(S_1, S_2, 0) = \begin{cases} 0, & \min(S_1, S_2) < K, \\ \frac{\mathrm{C}}{2}, & \min(S_1, S_2) = K, \\ \mathrm{C}, & \min(S_1, S_2) > K, \end{cases}$$

where K and C denote the strike price and the cash value of the Cash-or-Nothing option.

The change of variables $\hat{x} = Ke^{S_1}$, $\hat{y} = Ke^{S_2}$, and $\hat{t} = t$ is applied to (4.1), and it yields

$$\partial_{\hat{t}}\hat{u} + \left[-\frac{\sigma_1^2}{2}\partial_{\hat{x}}^2 - \rho\sigma_1\sigma_2\partial_{\hat{x}}\partial_{\hat{y}} - \frac{\sigma_2^2}{2}\partial_{\hat{y}}^2 - (r - \frac{\sigma_1^2}{2})\partial_{\hat{x}} - (r - \frac{\sigma_2^2}{2})\partial_{\hat{y}} \right] \hat{u} \quad (4.2)$$
$$+ r\hat{u} = 0.$$

The mixed derivative in (4.2) is removed by applying the coordinate transformation that is described in §2.2. The resulting equation,

$$\partial_t u + \left[-\frac{\sigma_1^2}{2}(1-\rho^2)\partial_x^2 - \frac{\sigma_2^2}{2}\partial_y^2 - \left\{ (r - \frac{\sigma_1^2}{2})(1 - \frac{\rho\sigma_1}{s_2}) \right\} \partial_x - (r - \frac{\sigma_2^2}{2})\partial_y \right] u \tag{4.3}$$

$$+ ru = 0$$

is approximated using the proposed finite difference scheme.

Since the proposed scheme is defined on a rectangular region, the computation region in (S_1, S_2) should be chosen in such a way that the coordinate transformation results in a rectangular region in (x, y) domain. A region in (S_1, S_2) domain is formed using the inverse of the function used in the coordinate transformation. A graphical illustration is given in Figure 4.1.

For constant parameters σ_1, σ_2, ρ, and r, (4.1) has the exact solution in the form,

$$v_{ex}(S_1, S_2, t) = \mathrm{C}e^{-rT} B\left(\frac{\log(S_1/K) + (r - 0.5\sigma_1^2)T}{\sigma_1\sqrt{T}}, \frac{\log(S_2/K) + (r - 0.5\sigma_2^2)T}{\sigma_2\sqrt{T}}, \rho \right),$$

where B is the bivariate normal cumulative distribution function. The error of the numerical solution is computed by comparing with the exact solution. Table 4.2 reports the error in l^∞-norm and the reduction rate when $\sigma_1 = \sigma_2 = 0.5$, $\rho = 0.5$, and $r = 0.03$. The fourth-order convergence is observed.

Note that since our purpose is to check the order of convergence of the finite-difference scheme, we impose the exact value on the boundary for the numerical solution not to be affected by the choice of boundary conditions.

(N_x, N_y)	N_t	l^∞-error	Reduction rate
(32, 32)	32	0.334E-04	
(64, 64)	128	0.206E-05	4.02
(128, 128)	512	0.129E-06	3.99

Table 4.2: Convergence result of Example 4.2

Example 4.3. The equity swap equation (1.1) is solved using the proposed fourth order compact scheme. The coefficients of the equation are given in

variable forms such that

$$\begin{aligned}
\sigma^r(t) &= 1+t, \qquad \sigma_2^S(y,t) = 1+y+t, \\
\rho &= 0.5, \qquad \kappa_t x = 2.0tx, \\
\zeta(x,y,t) &= x(x-1)+y(y-1)-t, \quad \gamma(x,y,t) = -x-y-t.
\end{aligned}$$

The zero forcing term (the right hand side) in (1.1) is replaced with a term generated by substituting u with the following function:

$$u_{ex}(x,y,t) = e^{-t}\sin\pi x \sin\pi y.$$

The coordinate transformation function $\phi(\hat{x},\hat{y},\hat{t})$ is written explicitly such that

$$\phi(\hat{x},\hat{y},\hat{t}) = \hat{x} - 0.5(1+\hat{t})\left(\ln(1+\hat{y}+\hat{t}) - \ln(1+\hat{t})\right).$$

The transformation eliminates the mixed derivative and results in an equation,

$$\begin{aligned}
\partial_t u(x,y,t) \;-\; & \left[a_1(t)\partial_x^2 + a_2(y,t)\partial_y^2\right. \\
& \left. + c_1(x,y,t)\partial_x + c_2(x,y,t)\partial_y + d(x,y,t)\right] u(x,y,t) = f(x,y,t)
\end{aligned}$$

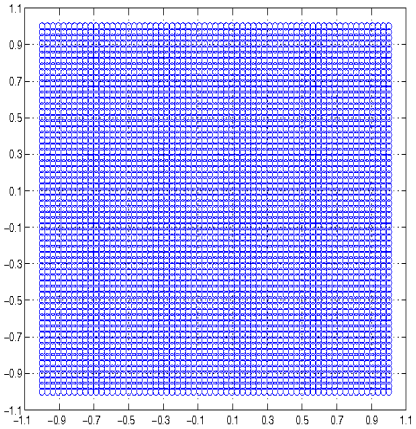
where the coefficients are defined as

$$\begin{aligned}
a_1(t) &= 0.375(1+t)^2, \\
a_2(y,t) &= 0.5(1+t)^2, \\
c_1(x,y,t) &= 0.5(1+t) - 2tx + 0.5\left\{x(1-x)+y(1-y)+t+1\right\}\frac{1+t}{1+y+t} \\
&\quad +0.5\left(\ln(1+y+t) - \ln(1+t)\right) - 0.5, \\
c_2(x,y,t) &= x(x-1)+y(y-1)-t, \\
d(x,y,t) &= -x-y-t.
\end{aligned}$$

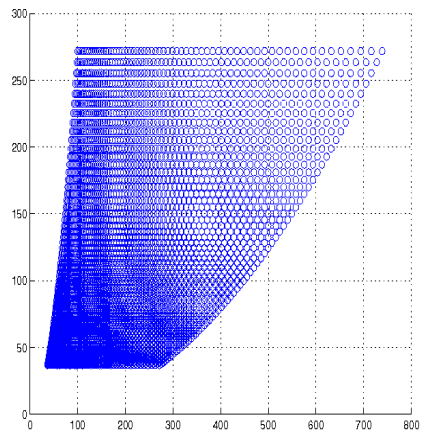
The computation results are reported in Table 4.3, and it confirms the fourth-order convergence.

(N_x, N_y)	N_t	l^∞-error	Reduction rate
(32, 32)	8	0.680E-03	
(64, 64)	32	0.423E-04	4.00
(128, 128)	128	0.273E-05	3.95

Table 4.3: Convergence result of Example 4.3



(a) A region in (x, y) domain



(b) A region in (S_1, S_2) domain

Figure 4.1: A rectangular region in (x, y) obtained from a region in (S_1, S_2)

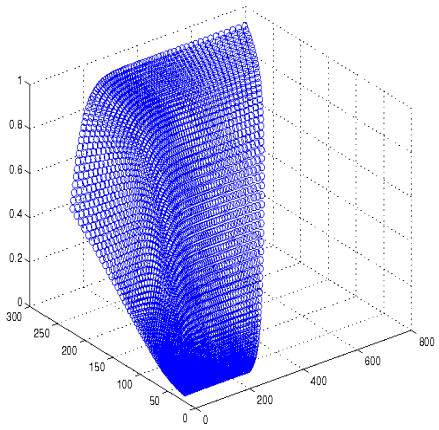


Figure 4.2: Numerical solution of Example 4.2

Chapter 5

Conclusions

In this thesis we applied fourth-order compact finite-difference scheme to the two-dimensional parabolic equation with variable coefficients and mixed derivatives. The key idea of developing high-order scheme is a transformation to remove mixed derivatives so that compactness is preserved.

A transformed equation simplifies to an unsteady convection-diffusion equation, and it can be easily discretized by a nine-point compact stencil with fourth-order convergence rate. Using that scheme and transformation, we proved fourth-order convergence with various numerical examples. The main advantages of this approach show less computations and more accurate results since this method reaches a reliable bound much faster than usual second-order approximations. Also we tried to impose variable coefficients in each coefficients of derivatives, so we can deal with more general cases in applications.

In conclusion, our high-order compact finite-difference scheme is very close to the real applications in which fast and accurate evaluation is critical. Our proposed fourth-order compact scheme will enhance the performance as well as provide highly accurate results.

Part II

Heterogeneous Computing with OpenCL

Chapter 1

Introduction

As computational resources are being developed in parallel, new parallel computing strategies enable us to utilize the full performance of system resources. GPU (Graphical Processing Unit) is a good example of massive parallel architecture, and there have been through studies on efficient algorithms for GPUs. Also CPUs (Central Processing Units) have evolved into multi–core architectures since the clock speed has been limited by cooling costs. Under these circumstances, even a normal personal computer has both a CPU and GPU, and these can perform enormous FLOPs (FLoating-point OPerations per second). Therefore, heterogeneous systems have emerged as a new paradigm for computational architectures because significantly increasing FLOPs cannot be achieved without heterogeneous computing skills. Although the heterogeneous parallelism for many of algorithms is still not an easy problem, many of future computations will demand computational powers as much as possible.

Considering scientific computations, focusing on heterogeneous computing is a natural target to improve traditional MPI (Message Passing Interface) based parallel algorithms. The most important factor in a heterogeneous system is load balancing since resources are not compatible with each other in a sense of architecture. In order to program for heterogeneous system, we use OpenCL (Open Computing Language), which is a standard heterogeneous computing language [31]. However, it still remains a difficult problem to balance workloads among computational resources.

In general, load balancing strategies are regarded as two categories.

- Task–wise load balancing
- Function–wise load balancing

At first, task–wise load balancing represents that each task can be assigned into various computational resources based on their computational characteristic. For instance, matrix multiplication task is appropriate for a GPU whereas simple vector addition takes the least time on a CPU. Thus, the dominance between computations and data transfers determines which resources are more suitable for. On the other hands, a function–wise load balancing deals with only one function. BLAS are good examples for this function–wise approach. Considering a vector addition, we assign some portions of vectors into various resources, so each resource carries out same computation.

Focusing on function-wise load balancing, we describe a load balancing problem between a CPU and GPU as a min–max problem for computational time of the CPU and GPU. If the CPU and GPU execute a job concurrently, the termination time is the maximum time between the CPU and the GPU. To minimize this maximum termination time is a natural approach for the heterogeneous load balancing problem. In addition, we estimated the performance of the CPU and GPU as functions in order to express the objective function for min–max model explicitly.

From previous studies, this min–max problem have been suggested in a distributed system as HEFT (Heterogeneous Earliest-Finish-Time) algorithm [17]. Also, the paper [34] proposed a fast portion of prediction to determine the portion of matrix for each device. This is almost similar with our goal, but we will estimate the performance of each resource so that conclude an ideal splitting ratio using benchmarked hardware parameters. And other recent researches emphasize heterogeneous computing [3, 8, 16].

In this thesis, we investigate how to balance loads between a multi–core CPU and a many–core GPU for BLAS Level 1 saxpy, BLAS Level 2 sgemv, and BLAS Level 3 sgemm. First of all, we developed a hypothesis to express loads for each computational resource as a function. And next, we verified this hypothesis by generating sample data both the CPU and GPU separately. Thus, we are able to compare this hypothetical estimation with real heterogeneous computing results. Finally, we analyze the result and conclude the goodness of fit with our model.

Chapter 2

OpenCL

We briefly review OpenCL to support our model analysis. OpenCL programs need platforms at first. The platform is a framework supplied by a vendor–specific OpenCL software development kit. Under this platform, vendor–specific computing devices are wrapped by corresponding OpenCL objects. The device objects can be made from GPUs, accelerators, or even CPUs. These various devices collaborate under a single platform or multiple platforms in terms of really heterogeneous computing. In the case of CPUs, the computing cores are shared by OpenCL programs on both devices and the host, which requires consideration on resource contention. Based on the platform and devices, context objects are made to contain computation-related objects like buffers, command-queues, kernels, events, and programs as a whole. Kernels are submitted into command-queues to execute functions on the devices.

We specifically want to address especially two topics: 1) Memory hierarchy and transfer, 2) Time Measurement. On the host, OpenCL programs manage vectors and matrices packed in one-dimensional arrays using C pointers. On devices, three memory spaces like global, local, and private memory exist. Usually, host memory and device memories reside on a distinct physical space, so that arrays on the host should be transferred into the device global memory before running a job. Most commonly, it is done by `EnqueueWriteBuffer` and `EnqueueReadBuffer`. For GPUs and accelerators, array transfer occurs through a PCI express bus. This accompanies latency and bandwidth of the bus in measuring computation performance. For CPUs, host memory and device memory are in the same physical space, we can make OpenCL buffer objects get the same

address with host arrays. This advantage offers better speed.

Function (kernel) execution on the devices is managed by the command-queue attached to each device. The queue usually has an in-order attribute that enforce a first-in, first-out timeline. Execution of kernels is profiled by event objects attached to command-queues. Events have four flags: *queued*, *submitted*, *start*, and *end* on the status of job execution. The first two states mean latency for execution including software call stack establishment, delay for completion of a previous kernel. The next two states measure a net running time of a kernel. However, the event variable has additional overheads to enable profiling with $10\mu s$ [5]. Moreover, time measurement with the event variable cannot reflect heterogeneous computing time since it only measures one `clEnqueue*` job at once. Instead, AMD SDK [7] offers a reliable nano-second timer, so this measures the heterogeneous computing time by wrapping all `clEnqueue*` operations. In next chapter, we will deal with some implementation issues concerning with those time measurements.

Chapter 3

Implementation Issues

Heterogeneous computing between a CPU and GPU is emerging rapidly. OpenCL have been designed to utilize heterogeneous system, but it is not fully reliable since some strange behaviors were observed. As briefly mentioned about previous chapter, if someone wants to use a multi-core CPU, which manages OpenCL scheduler, full performance of the CPU may not be achieved properly. In this chapter, we will describe some of those strange situations as well as suggest alternative ways to avoid those. We generated all sampling data and benchmarks under following environment Table 3.1.

Parameters	CPU (AMD FX 8120)	GPU (ATI HD 7950)
Number of compute units	8	32
Core clock speed	3.1 GHz	0.8 GHz
Global memory size	16 GB	3 GB
Operating system	Windows 7	Windows 7
OpenCL SDK version	2.8	2.8
Peak Performance (float)	37.2 GFlops	2867 GFlops
Global memory bandwidth	21.3 GB/s	240 GB/s

Table 3.1: Heterogeneous Computing Environment

3.1 Concurrency in Heterogeneous Computing

While gathering statistics for heterogeneous computing results, we found that a maximum time between the CPU and GPU at each moving variable k was not observed, where the value k assigns k amount of operations to the CPU and $n - k$ to the GPU with fixed n variable. Especially, this occurs when the CPU computes more than the GPU as following Figure 3.1 shows.

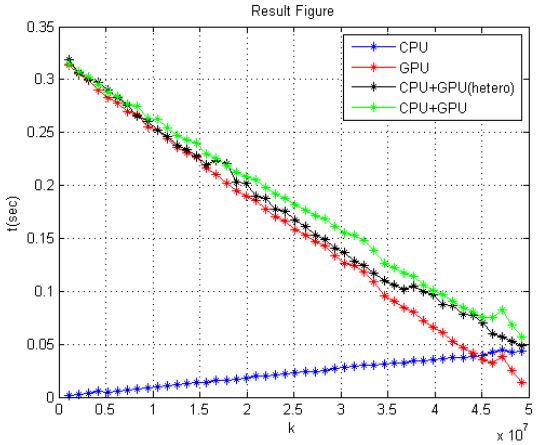
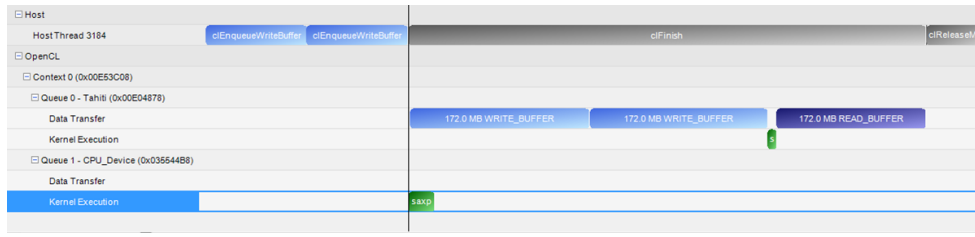
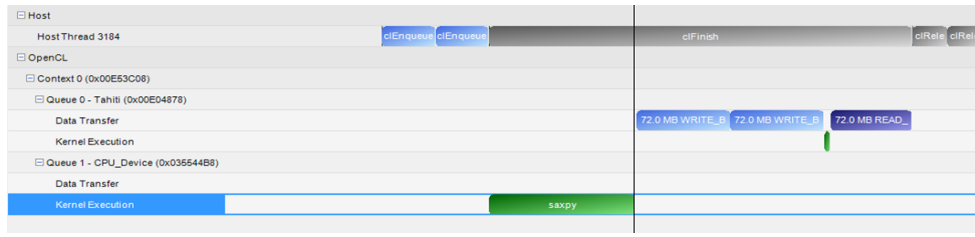


Figure 3.1: Heterogeneous computing results: 8–core CPU

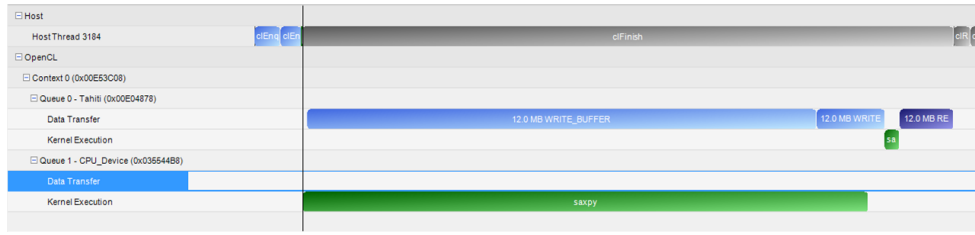
The blue line shows the taken time by the CPU with full 8–core from the Table 3.1. And the red line means the GPU computing time. Notice that the black line is heterogeneous computing results whereas the green line is just sum of the blue and red lines. Around $k = 3.5 \times 10^7$, unknown overheads were started to occur. We expected the black line follows the maximum value between the red and blue lines, but the black line reaches the green line. This implies that the heterogeneous execution is serialized for some reasons. Fortunately, we have AMD APP profiler [6] that analyze the time line for the given execution. Following figures are profiling results and contains strange behaviors.



(a) Working properly



(b) Execution serialized



(c) CPU holds the PCI express transfer

Figure 3.2: Profiling Results: 8–core CPU

Notice that if the portion of the CPU execution gets larger, total execution takes more time. In other words, the heterogeneous computing performance gets lower when the CPU, which schedules OpenCL instruction for both the CPU and GPU, occupies full–core computation. Using the OpenCL function `clCreateSubDevice`, we are able to split the 8–core CPU into valid sub-CPUs. We divided this CPU evenly, so that becomes two 4–core CPUs. Again, we generated a heterogeneous computing result using one 4–core CPU same with Figure 3.1 and the profiling results same with Figure 3.2.

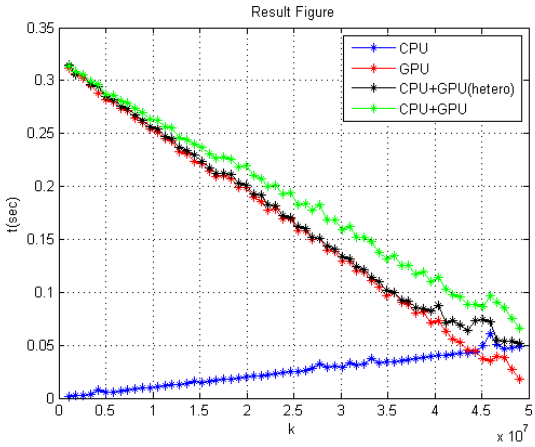
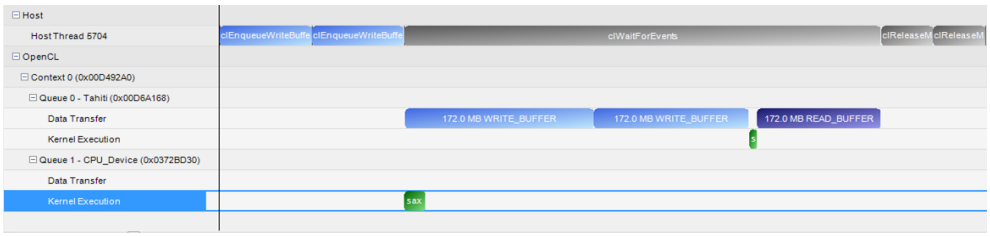
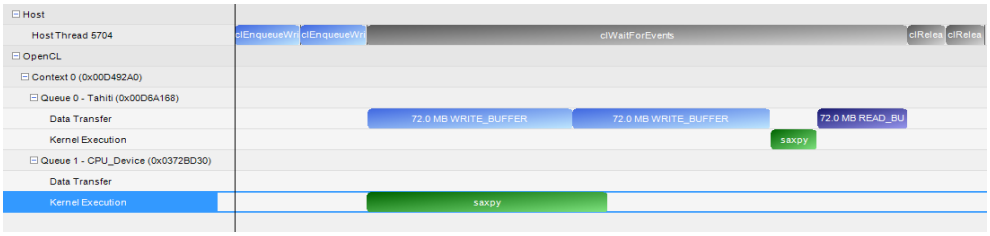


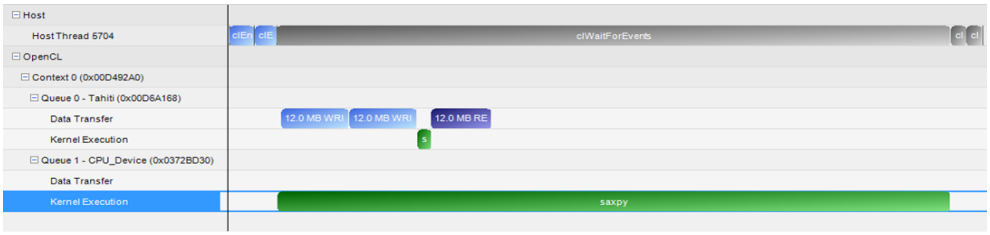
Figure 3.3: Heterogeneous computing results: 4–core CPU



(a) Working properly



(b) Working properly



(c) Working properly

Figure 3.4: Profiling Results: 4–core CPU

Comparing Figure 3.3 with Figure 3.1, we see that the 4–core result is more reliable but there still exists some overheads around the cross section point between the blue and red lines. The profiling results in Figure 3.4 show that concurrent executions between the CPU and GPU were observed well in contrast to Figure 3.2. However, there will be a trade-off using 4–core rather than 8–core. Following Figure 3.5 shows differences between execution time for saxpy operations.

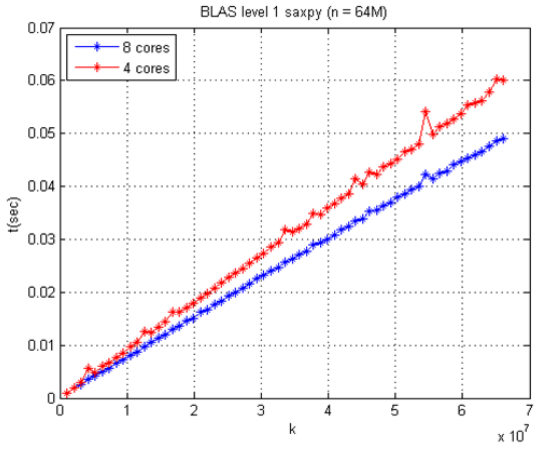


Figure 3.5: Differences between 4–core and 8–core execution time

Definitely 8–core execution is faster than 4–core about 10 percent. However, it is more persuasive to use sub-CPU if the CPU resource contention matters for computational performances. So we will use the 4–core sub-CPU throughout this paper. In next subsection, we keep investigating to figure out overheads around the cross section point.

3.2 CPU Parking Protocol

Windows 7 has a special protocol for cooling CPU cores. Microsoft does not fully state advantages or disadvantages of this parking protocol, but by default some of CPU cores are parked and switchings are also observed between normal cores and parked cores. From web articles [1,2], parking protocol aims at a lower power consumption, but this will give some unpredictable overheads when the multi-core CPU runs heavy jobs. Fortunately, users can customize registries on Windows 7, so that releases the core parking strategy. In other words, we can represent it as Figure 3.6.

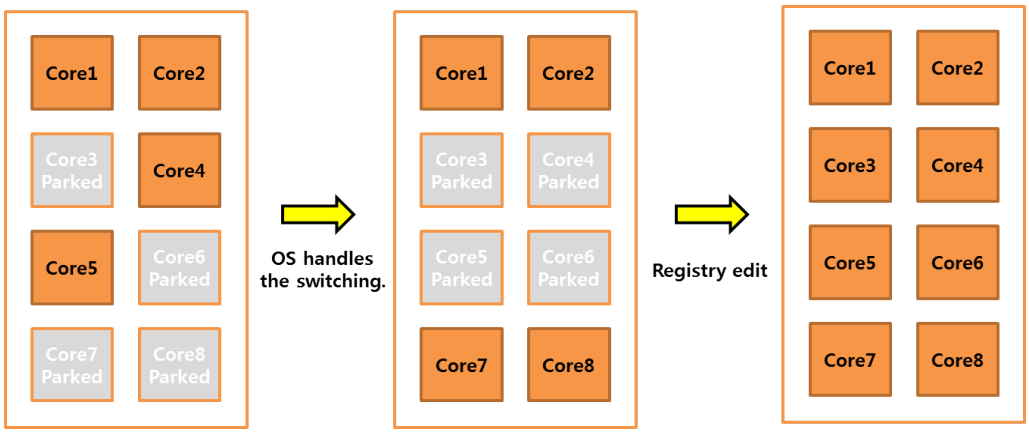


Figure 3.6: CPU core parking

The main reason why this core parking matters is that the heavier computation runs, the more irregular results are observed. We tested BLAS Level 1 saxpy algorithm, which is scalar multiple of vector addition. The variable k varies from 1048576(=1M) to 67108864(=64M) with step size 1048576(=1M). The number of cores used in this example is 4–cores from the 8–cores CPU in order to exclude resource contentions as we suggested in SS3.1.

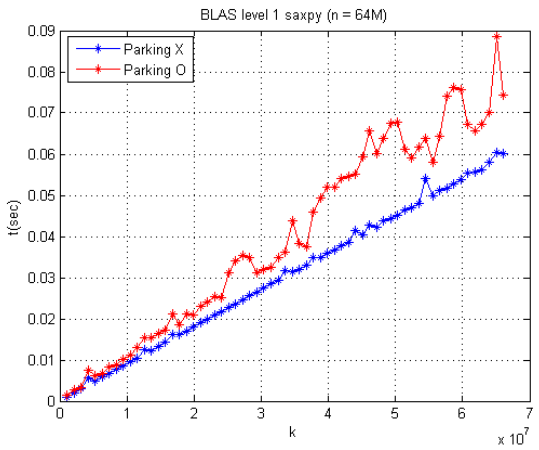


Figure 3.7: Sampling results between enabling and disabling CPU parking

In Figure 3.7, the blue line shows disabled CPU parking results while the

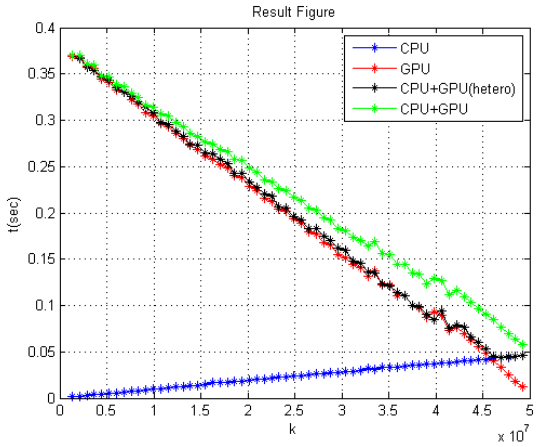


Figure 3.8: Heterogenous computing result : 4–core and disabled CPU parking

red line shows enabled CPU parking. From this figure, we could conclude that invoking parked cores has unpredictable overheads. Of course, there is a trade-off disabling CPU parking. Each CPU core is easily heated, so cooling fans run almost maximum RPM which consumes more electricity. Notice that the blue line shows consistent growth whereas the red line has relatively more oscillations. Therefore, disabling CPU parking is a good choice to generate robust sampling data although it consumes more powers. Using this observation, we test an example Figure 3.8 same with Figure 3.1 and Figure 3.3. This Figure 3.8 shows that our expectations were satisfied since the black line follows the maximum value of the blue and red lines.

Chapter 4

Modeling

There are many ways to measure the loads. However, we are primarily interested in how fast computational resources execute given loads. This is nothing but FLOPs that is a powerful measurement for computational power. Given a deterministic load, we can then calculate the FLOPs measuring the time to completion. Therefore, it is apparent that we must first construct time to completion models for each resource.

First, we assume that all the loads are consists of input n. For instance, saxpy, single precision computation of $ax + y$, operation with vector length n has n floating point operations, $2n+1$ times 4 bytes amount of memory reading, and n times 4 bytes amount of memory writing.

Then, one can estimate how much time will be taken for the computation. Of course, it is not that easy to evaluate exactly, but we know that how a computation works with hardware features. For example, memory bandwidth impacts how fast target data can be fetched in private memory. Also, the theoretical FLOPs for each computational resources measures how much time is spent for given number of floating point operations. These two factors have major effects on the computational time, and additional overheads will determine latencies. The following definition states the parameterized time functions with input variables for both a CPU and GPU.

Definition 4.1. For given input variable n, the following cost functions evaluate how long the computation takes.

1. $c(n) = c(n; B_{CPU}, F_{CPU}, O_{CPU})$: time taken by CPU

2. $g(n) = g(n; B_{PCI,R}, B_{PCI,W}, B_{GPU}, O_{GPU})$: time taken by GPU

where each parameter stands for

- subscripts CPU and GPU
  - B : memory bandwidth
  - F : theoretical FLOPs
  - K : OpenCL kernel launching overhead
  - O : overheads
- PCI
  - $B_{PCI,W}$: PCI Writing bandwidth (Host to Device)
  - $B_{PCI,R}$: PCI Reading bandwidth (Device to Host)

One advantage of heterogeneous computing is that one can utilize many different resources concurrently. Obviously, independent jobs should be assigned at each resource to maximize the utility. On the other hand, if one tries to solve a problem using one CPU and one GPU, for instance, one need to split the amount of jobs to each to minimize the total amount of time to complete the job. Let us call the split ratio as α such that αn amount of input will be assigned to the CPU and $(1 - \alpha)n$ to the GPU. In order to clarify this concept, we will frame a hypothetical model as follows. Find a splitting point k^* which is the solution of the following minimax problem:

$$J(k^*; n) = \min_{k \in [0,n]} J(k; n), \tag{4.1}$$

where $J(k; n) = \max(c(k), g(n - k))$, from which $\alpha = \alpha(n)$ is defined by

$$\alpha(n) = \frac{k^*}{n}. \tag{4.2}$$

This argument comes from the simple fact that for a fixed n amount of jobs if a GPU computes faster than a CPU for some splitting point k, then the GPU still has to wait for the CPU to finish, and vice versa. Therefore, we need to take the maximum time of $c(k)$ and $g(n-k)$. From this intuition, we can derive a successful model by measuring the minimum of those maximum values.

One more important feature could be realized if we recall the nature of $c(n)$ and $g(n)$. These time functions are mainly depending on time complexity, thus

we are going to consider their polynomial complexities. Therefore the functions $c(n)$ and $g(n)$ will be assumed as polynomials with their domains restricted to $[0, n]$, from which an exact solution to (4.1) can be derived without difficulty.

First, notice that k^* should be in the following admissible set

$$\begin{aligned} \mathcal{A} &= \{0, n\} \cup \{k \in (0, n) \mid c(k) = g(n-k)\} \\ &\cup \{k \in (0, n) \mid \frac{dc}{dk}(k) = 0 \text{ and } \frac{d^2c}{dk^2}(k) < 0\} \\ &\cup \{k \in (0, n) \mid \frac{dg}{dk}(n-k) = 0 \text{ and } \frac{d^2g}{dk^2}(n-k) < 0\}. \end{aligned} \tag{4.3}$$

The meaning of the above admissible set $\mathcal{A}$ can be understood as follows. There are some rate cases where the jobs may be complete faster in solving by dedicated to either a CPU or a GPU rather than in solving them after distributing to both computing resources. In this situation, $k^* \in \{0, n\}$. Otherwise, one has to check the cross section points, based on which the domain is divided into subintervals. In each subinterval, the maximum of functions $c(k)$ and $g(n-k)$ can occur at either at one of the end points or at an interior point. The interior maximum can be found by using the first and second derivative tests. Observe that all these candidates for $\mathcal{A}$ are finite, whose minimum is the solution of (4.1).

4.1 Performance Estimations

For instance, BLAS level 1 algorithms have clearly linear time complexity so that we can verify our model successfully. We also recognized that the most time consuming parts are memory transfers and computations. The input parameter n varies not only the amount of memory to be transferred but the number of operations. Therefore the leading coefficient of polynomial time function tells us whether this algorithm is data transfer or computation dominant. Our model simplified the procedure to construct time functions as follows.

$$\text{time} = \text{computation time} + \text{data transfer time} + \text{overheads and latency}$$

Overheads and latency may be negligible depending on the input variable n, but we dealt with as constant variables since computation and data transfer time will have major effects on total consumed time. Using Definition 4.1, we are able to construct the time functions. The saxpy operation is a BLAS Level 1

algorithm as follows.

$$y = ax + y, \quad a \in \mathbb{R} \quad \text{and} \quad x, y \in \mathbb{R}^n$$

We start from a very basic step. Theoretical FLOPs can be calculated by

$F =$ number of cores $\times$ maximum clock speed $\times$ number of operations per cycle.

Here, the operation represents multiplication and addition at once, shortly one floating point operation. The saxpy algorithm has n floating–point operations and the computation will take the following amount of time.

$$\frac{n \text{ (operations)}}{F \text{ (operations / second)}}. \tag{4.4}$$

Now, we will calculate memory transfer. We assume a configuration where the GPU is connected through PCI express as a device, and the CPU uses host memory as global memory in OpenCL concepts. In the host memory, the vector x and y are already allocated and assigned appropriate random values. Therefore, vector x and y are firstly copied from global memory to private memory, and then resulting y vector is written back from private memory to global memory. Notice that we have additional reading and writing through the PCI express in the GPU case. In summary,

$$\frac{2 \times 4 \; bytes \times n}{B_{CPU}} + \frac{4 \; bytes \times n}{B_{CPU}} \tag{4.5}$$

for the CPU case and for the GPU

$$\frac{2 \times 4 \; bytes \times n}{B_{PCI,W}} + \frac{4 \; bytes \times n}{B_{PCI,R}} + \frac{2 \times 4 \; bytes \times n}{B_{GPU}} + \frac{4 \; bytes \times n}{B_{GPU}}. \tag{4.6}$$

By adding those two main factors and overheads, we can develop time functions $c(n)$ and $g(n)$ as follows

$$c(n) = \frac{12n}{B_{CPU}} + \frac{n}{F_{CPU}} + O_{CPU}, \tag{4.7}$$

$$g(n) = \frac{8n}{B_{PCI,W}} + \frac{4n}{B_{PCI,R}} + \frac{12n}{B_{GPU}} + \frac{n}{F_{GPU}} + O_{GPU}. \tag{4.8}$$

where each parameters are given in Appendix A. We will call those parameters

reference parameters. To evaluate reference parameters, we collect actual bandwidths and FLOPs generated from sampling statistics. For instance, reference bandwidths for saxpy operation such as B_{CPU}, B_{GPU}, $B_{PCI,W}$, and $B_{PCI,R}$ are averaged values of sampling bandwidths. Let us focus on B_{GPU}. We can make a kernel function that only executes two writings and one reading with variable n. Then, we estimated the gradient part from the linear regression and evaluate B^n_{GPU} using this gradient. Therefore, the reference parameter B_{GPU} is defined by

$$B_{GPU} = \frac{B^{64M}_{GPU} + B^{56M}_{GPU} + B^{48M}_{GPU} + ...}{\text{the number of sampling statistics}}. \tag{4.9}$$

On the other hands, the FLOPs F_{CPU} and F_{GPU} is actually given by theoretical amount of FLOPs. Once we get reference bandwidths, we are able to evaluate reference FLOPs by subtracting taken time by the saxpy operation with memory transfer times. However, in cases of sgemv and sgemm, there exist differences between subtracted values and theoretical FLOPs since simple assignments, from global memory to private memory or vice versa, cannot fully reflect cache memory behaviors or dedicate procedures for executions. Thus, we used subtracted values to estimate reference FLOPs for sgemv and sgemm cases.

Especially the PCI express bandwidth $B_{PCI,W}$ and $B_{PCI,R}$ were estimated with similar manners. However, those values are really far from the theoretical performance. So we did investigate those in next chapter.

4.2 PCI express Bandwidth

The theoretical bandwidth for PCI express 2.0 x16 is 8GB/s. But this cannot be reached from the fact that in order to send and receive meaningful 64bytes data, there should be additional 20 bytes for checking errors. So, the performance decreases 76 percent from the theoretical values [11]. Usual benchmarks of PCI express 2.0 x16 are at least 5.5 GB/s, but this parameter is also highly dependent on the mother board. The benchmarked machine is nothing but simple personal computer, so we may not expect full performances. Therefore it is natural to use a bandwith coming from benchmarking. In Section 2, we introduced an event variable to measure the time. Again, this event variable cannot fully reflect real transfer times since it does not contain the latency from host memory to PCI memory. However, measuring with event variable is more closed to theoretical bandwidth, and therefore we can clarify where degradations occur. Following

Figure 4.1 shows latencies about memory transfers.

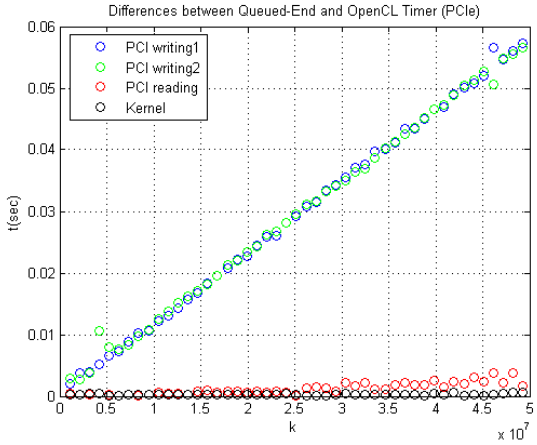


Figure 4.1: Differences between event variable and AMD timer

This figure was generated by the differences between timing measurement methods. Of course, since AMD timer contains all the latencies, the time measured with an event variable from *queued* to *end* status is smaller than the time measured with AMD timer. These subtracted values imply how much degradations occurs about PCI express bandwidth. The PCI writing operation has linear overhead which can be analyzed additional costs moving from host memory to PCI memory and the PCI reading operation as well. We will call this as minor bandwidths, so this needs to be subtracted from the bandwidth measured by event variables. We report all those parameters in Appendix A.

Chapter 5

Results

Example 5.1. Consider single precision scalar multiplication and vector addition, shortly saxpy. The saxpy operation is a BLAS level 1 algorithm and we are going to find the splitting ratio $\alpha(n)$.

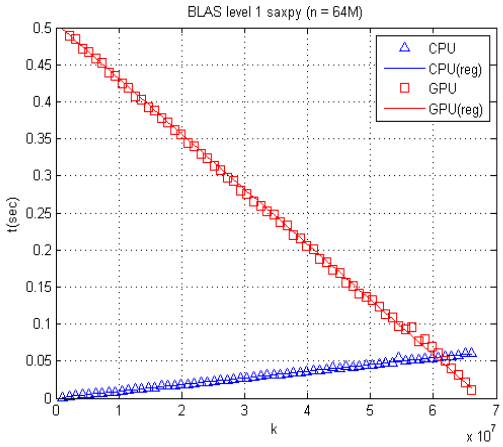
$$y = ax + y, \quad a \in \mathbb{R} \quad \text{and} \quad x, y \in \mathbb{R}^n$$

The performance will be estimated by following function.

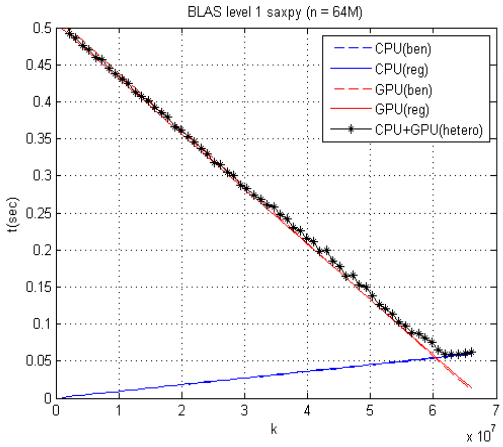
$$\begin{aligned} c(k) &= \frac{12k}{B_{CPU}} + \frac{k}{F_{CPU}} + O_{CPU}, \\ g(k) &= \frac{8k}{B_{PCI,W}} + \frac{4k}{B_{PCI,R}} + \frac{12k}{B_{GPU}} + \frac{k}{F_{GPU}} + O_{GPU} \end{aligned} \tag{5.1}$$

where each parameters are in Table A.2.

We first give a result with $n = 67108864$.



(a) Regression



(b) Comparing with heterogeneous computing

Figure 5.1: The result of Example 5.1 with n=67108864

Figure 5.1(a) shows a linear regression with collection of sampling statistics. On the other hand, Figure 5.1(b) has solid, dashed, and marked lines. The solid blue and red lines are coming from Figure 5.1(a), and the dashed blue and red lines mean functions (5.1) with referenced parameters. Finally, the marked black line states the heterogeneous computing results. We can justify the goodness of estimation by comparing cross section points and the minimum value heterogeneous computing results. We did make sampling data with various

n, so the following Figure 5.4 shows splitting ratio $\alpha(n)$.

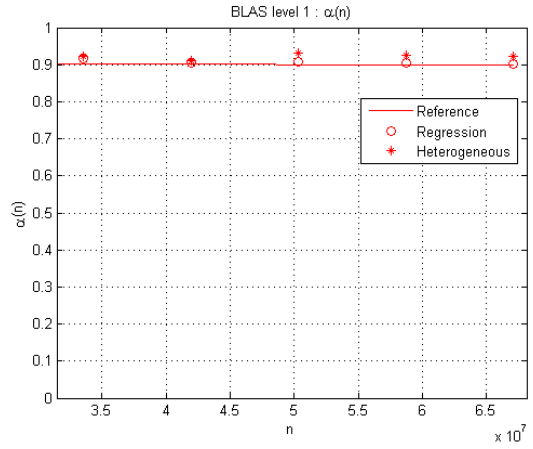


Figure 5.2: The splitting ratio $\alpha(n)$ for Example 5.1

Since we have explicit formula (5.1), we can express $\alpha(n)$ as a function. The cross section point from the regressed linear function and the minimum value of the heterogeneous computing result are in 5 percent difference with reference parameter functions.

Example 5.2. Consider single precision matrix vector multiplication, shortly sgemv. The sgemv operation is a BLAS level 2 algorithm and we are going to find the splitting ratio $\alpha(n)$.

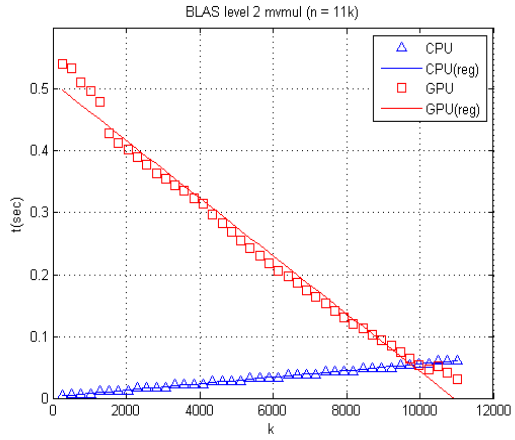
$$y = Ax, \quad A \in \mathbb{R}^{n\times n} \quad \text{and} \quad x, y \in \mathbb{R}^n$$

The performance will be estimated by following function with given n.

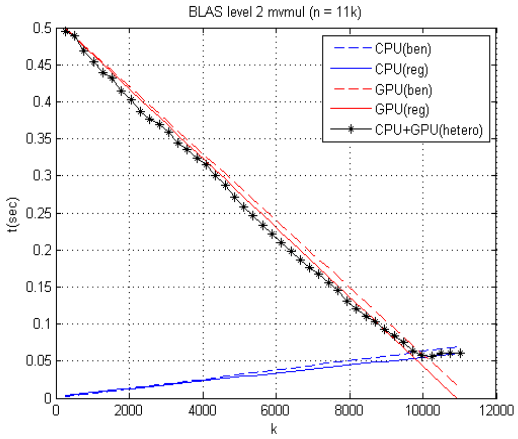
$$\begin{aligned} c(k) &= \frac{4(n+2)k}{B_{CPU}} + \frac{nk}{F_{CPU}} + O_{CPU}, \\ g(k) &= \frac{4(n+1)k}{B_{PCI,W}} + \frac{4k}{B_{PCI,R}} + \frac{4(n+2)k}{B_{GPU}} + \frac{nk}{F_{GPU}} + O_{GPU} \end{aligned} \tag{5.2}$$

where each parameters are in Table A.3.

We give a result with $n = 11264$.



(a) Regression



(b) Comparing with heterogeneous computing

Figure 5.3: The result of Example 5.2 with n=11264

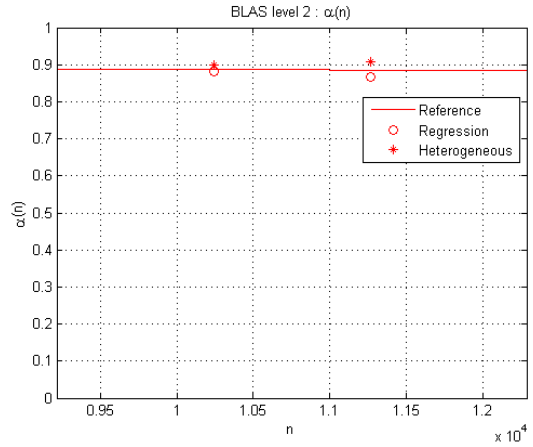


Figure 5.4: The splitting ratio $\alpha(n)$ for Example 5.2

Example 5.3. Consider single precision matrix matrix multiplication, shortly sgemm. The sgemn operation is a BLAS level 3 algorithm and we are going to find the splitting ratio $\alpha(n)$.

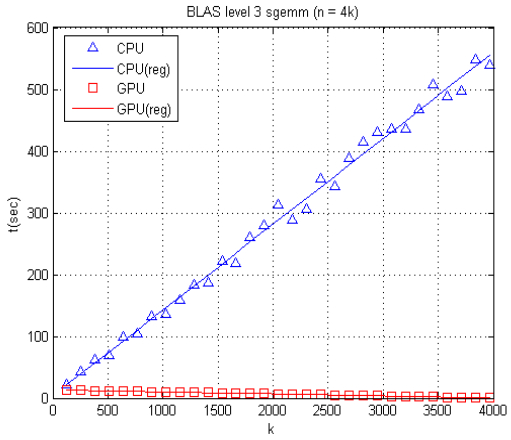
$$C = AB, \quad A, B, \text{ and } C \in \mathbb{R}^{n \times n}$$

The performance will be estimated by following function with given n.

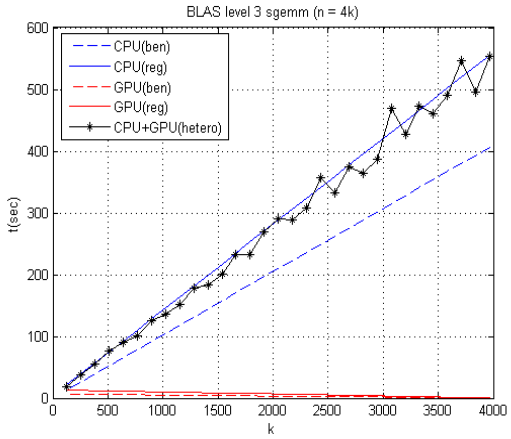
$$\begin{aligned} c(k) &= \frac{12nk}{B_{CPU}} + \frac{n^2k}{F_{CPU}} + \frac{4n^2}{B_{CPU}} + O_{CPU}, \\ g(k) &= \frac{8nk}{B_{PCI,W}} + \frac{4nk}{B_{PCI,R}} + \frac{12nk}{B_{GPU}} + \frac{n^2k}{F_{GPU}} + \frac{4n^2}{B_{PCI,W}} + \frac{4n^2}{B_{GPU}} + O_{GPU} \end{aligned} \tag{5.3}$$

where each parameters are in Table A.4.

We give a result with $n = 4096$.



(a) Regression



(b) Comparing with heterogeneous computing

Figure 5.5: The result of Example 5.3 with n=4096

We can conclude that this example is more appropriate to the GPU. Even though the kernel function for each device was not optimized, the GPU has more powerful potential to speed up.

Chapter 6

Conclusions

We have investigated how to balance given jobs in a heterogeneous system. The major approaches were to model load balancing features and to implement successful estimations in terms of time functions with a CPU and GPU. Notice that anticipating time taken by each resource is the core of obtaining exact splitting ratio α. However, the more complex a given algorithm, the more difficult it becomes to evaluate the time function. Therefore we gave simple examples in order to control those difficulties and to clarify the results.

We can conclude that our min–max model gives a successful splitting ratio for arbitrary n. The differences between reference and regressed parameters are well bounded, so we can expect a good load balancer using only reference parameters. Although regressed parameters from sample data will be better choice, sampling data will take too much time to be practical in application. However, it still remains a hard problem if a given algorithm is more complex.

Appendix A

Hardware Parameters

Parameters	PCI writing	Parameters	PCI reading
Bandwidth	2.686GB/s	Bandwidth	3.353 GB/s
Minor Bandwidth	1.532 GB/s	Minor Bandwidth	0.050 GB/s

Table A.1: Reference parameters: PCI express 2.0 x16

Parameters	CPU	Parameters	GPU
F_{CPU}	37.2 GFLOPs	F_{GPU}	2867 GFLOPs
B_{CPU}	13.527 GB/s	B_{GPU}	149.377 GB/s
O_{CPU}	3.528E-04 sec	O_{GPU}	2.527E-03 sec

Table A.2: Reference parameters: CPU and GPU (Example 5.1)

Parameters	CPU	Parameters	GPU
F_{CPU}	3.69 GFLOPs	F_{GPU}	2867 GFLOPs
B_{CPU}	13.527 GB/s	B_{GPU}	4.757 GB/s
O_{CPU}	3.528E-04 sec	O_{GPU}	2.527E-03 sec

Table A.3: Reference parameters: CPU and GPU (Example 5.2)

Parameters	CPU	Parameters	GPU
F_{CPU}	0.164 GFLOPs	F_{GPU}	11.0 GFLOPs
B_{CPU}	1.654 GB/s	B_{GPU}	2.132 GB/s
O_{CPU}	3.528E-04 sec	O_{GPU}	2.527E-03 sec

Table A.4: Reference parameters: CPU and GPU (Example 5.3)

Bibliography

[1] Enable or disable cpu core parking in windows 7. http://bitsum.com/about_cpu_core_parking.php.

[2] What is cpu parking? http://www.thewindowsclub.com/enable-disable-core-parking-window.

[3] A. Brodtkorba, C. Dyken, T. R. H. J. M. H., and Storaasli, O. O. State-of-the-art in heterogeneous computing. *Scientific Programming 18*, 1 (2010), 1–33.

[4] Achdou, Y., and Pironneau, O. *Computational methods for option pricing*. SIAM, Philadelphia, 2005.

[5] Advanced Micro Devices, Inc. *AMD Accelerated Parallel Processing OpenCL Programming Guide*, 2013.

[6] Advanced Micro Devices, Inc. *AMD APP Profiler*, 2013.

[7] Advanced Micro Devices, Inc. *AMD APP Software Development Toolkit Version 2.8*, 2013.

[8] Augonnet, C. *Scheduling Tasks over Multicore machinces enhanced with Accelerators: a Runtime System's Perspective*. PhD thesis, University Bordeaux 1, 2011.

[9] Brigo, D., and Mercurio, F. *Interest Rate Models – Theory and Practice With Smile, Inflation and Credit*, second ed. Springer Finance. Springer Verlag, Berlin Heidenlberg, 2006.

[10] Chance, D. M. Equity swaps and equity investing. *The Journal of Alternative Investments Summer 7*, 1 (2004), 75–97.

[11] Coleman, J., and Taylor, P. Hardware level io benchmarking of pci express. Tech. Rep. 321071, Intel Corporation, 2008.

[12] Duffy, D. J. *Finite Difference Methods in Financial Engineering.* John Wiley & Sons, Ltd., Chichester, West Sussex, 2006.

[13] Filipović, D. *Interest rate models.* Skriptum, Universität München, 2006.

[14] Geiser, J. Higer-order difference and higher-order splitting methods for 2D parabolic problems with mixed derivatives. *Inter. Math. Forum. 67* (2007), 3339–3350.

[15] Gibson, R., Lhabitant, F., Pistre, N., and Talay, D. Interest rate model risk: an overview. *Journal of risk 1* (1998), 37–62.

[16] Grewe, D., and O'Boyle, M. F. A static task partitioning approach for heterogeneous systems using opencl. *Compiler Construction*, 2011 (286–305).

[17] H. Topcuoglu, S. H., and Wu, M. Performance-effective and low-complexity task scheduling for hetetrogeneous computing. *IEEE Transactions on Parallel And Distributed Systems 13*, 3 (2002), 260–274.

[18] Hull, J. *Options, Futures, and Other Derivative Securities*, fourth ed. Prentice-Hall, Englewood-Cliff, N.J., 2000.

[19] Hull, J., and White, A. Pricing interest-rate derivative securities. *The Review of Financial Studies 3*, 4 (1990), 573–592.

[20] Karaa, S. High-order difference scheme for 2D elliptic and parabolic problems with mixed derivatives. *Numer. Methods Partial Differential Equations 23*, 2 (2007), 366–378.

[21] Karaa, S. A high-order ADI method for parabolic problems with variable coefficients. *Inter. J. Comp. Math. 86* (2009), 109–120.

[22] Karaa, S. Two-level compact implicit schemes for three-dimensional parabolic problems. *Comp. Math. with Applic. 58* (2009), 257–263.

[23] Karaa, S., and Zhang, J. High order ADI method for solving unsteady convection-diffusion problems. *J. Comput. Phys. 198* (2004), 1–9.

[24] Lesnoewki, A. Chapter 5. Short rate models. *Course on Interest Rate and Credit Models* (Spring 2007). http://www.math.nyu.edu/ alberts/spring07/index.html.

[25] Li, J., Chen, Y., and Liu, G. High-order compact ADI methods for parabolic equations. *Comp. Math. with Applic. 52* (2006), 1343–1356.

[26] Ma, Y., and Ge, Y. A high order finite difference method with Richardson extrapolation for 3D convection diffusion equation. *App. Math. Comp. 215* (2010), 3408–3417.

[27] Mercurio, F. Interest rates and the credit crunch: New formulas and market models (February 5, 2009). Tech. Rep. Bloomberg Portfolio Research Paper No. 2010-01-FRONTIERS, 2009. Available at SSRN: http://ssrn.com/abstract=1332205.

[28] O. Brockhaus, A. Ferraris, C. G. D. L. R. M., and Overhaus, M. *Modelling and hedging equity derivatives.* Risk, 1999.

[29] Overhaus, M., Bermudez, A., Buehler, H., Ferraris, A., Jordinson, C., and Lamnouar, A. *Equity hybrid derivatives.* John Wiley & Sons Inc., Hoboken, New Jersey, 2007.

[30] Overhaus, M., Ferraris, A., Knudsen, T., Mao, F., Nguyen-Ngoc, L., and Schindlmayr, G. *Equity derivatives: theory and applications*, vol. 106. Wiley, 2011.

[31] P. Du, R. Weber, P. L. S. T. G. P., and Dongarra, J. From cuda to opencl: Towards a performance–portable solution for multi–platform gpu programming. *Parallel Comput. 38*, 8 (2012).

[32] Qin, J. The new alternating directional implicit difference methods for solving three-dimensional parabolic equations. *App. Math. Modelling. 34* (2010), 890–897.

[33] Rebonato, R. *Modern Pricing of Interest-Rate Derivatives.* Princeton University Press, 2002.

[34] S. Ohshima, K. K., and Yuba, T. Parallel precessing of matrix multiplication in a cpu and gpu heterogeneous environment. *High Performance Computing for Computational Science* (2007), 305–318.

[35] Spotz, W., and Carey, G. Extension of high-order compact schemes to time-dependent problems. *Numer. Methods Partial Differential Equations 17* (2001), 657–672.

[36] Strikwerda, J. C. *Finite Difference Schemes and Partial Differential Equations*, 2nd ed. SIAM, 2004.

[37] Tian, Z. A rational high-order compact ADI method for unsteady convenction-diffusion equations. *Comp. Phys. Communication 182* (2011), 649–662.

[38] Tian, Z., and Ge, Y. A forth-order compact ADI method for solving tow-dimensional unsteady convection–diffusion equation. *J. Comput. Appl. Math. 198* (2005), 268–286.

[39] Wilmott, P., Dewynne, J., and Howison, S. *Option Pricing: Mathematical Models and Computation*. Oxford Financial Press, 1994.

초록

Equity Swap에 대한 고차 유한차분법과 OpenCL을 이용한 Heterogeneous 컴퓨팅

추형석
협동과정 계산과학전공
대학원
서울대학교

본 학위 논문에서는 Equity 스왑 모델에 대한 4차 수렴 유한차분법을 제안하였다. 특히 Equity 스왑 모델은 시간과 공간에 종속하는 계수들을 가지고 있기 때문에, 4차 수렴 유한차분법을 유도하기 위하여 특별한 좌표 변환을 고려하였다. 이 좌표 변환은 편미분 방정식에서 교차미분을 제거하는 것으로, 여러 예제들을 통해 그 수렴성을 검증하였다.

대부분의 선형해법들은 BLAS 알고리즘을 기반하여 구성되어있기 때문에, CPU와 GPU를 사용하여 BLAS를 병렬화 하는 연구를 수행하였다. 이것은 CPU와 GPU에 어떻게 작업을 분배할 것인가의 문제로 귀결되고, 분배하는 지점은 각 계산자원에서 소요되는 계산시간의 최소–최대 문제로 나타낼 수 있다. CPU와 GPU에서 특정 BLAS를 계산하는데 걸리는 시간을 다항함수의 형태로 예측함으로써, 최소–최대 문제와 실제 계산결과를 비교 분석하였다.

주요어 : 고차 유한차분법, Equity 스왑,
Hetergeneous 컴퓨팅, OpenCL, BLAS,
최소–최대 문제
학 번 : 2009-20453